50376
1984
315

université des sciences et techniques de lille flandres artois

N° d'ordre : 627

THÈSE

présentée à

L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE FLANDRES ARTOIS

pour obtenir le titre de

DOCTEUR ES SCIENCES MATHEMATIQUES

par

Michel MERIAUX

BIBLIOTHEQUE INTERUNIVERSITAIRE SECTION DE SCIENCES LILLE

CONTRIBUTIONS A L'IMAGERIE INFORMATIQUE : ASPECTS ALGORITHMIQUES ET ARCHITECTURAUX

Soutenue le 3 Juillet 1984 devant la Commission d'Examen

Membres du Jury :

M. DAUCHET	Président
V. CORDONNIER	Rapporteur
M. LUCAS	Rapporteur
F. MARTINEZ	Rapporteur
C. GIRAULT	Examinateur
G. GAILLAT	Examinateur
A. RECOQUE	Examinateur

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.F.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.

Je tiens ici à exprimer ma profonde reconnaissance à toutes les personnes qui, d'une manière ou d'une autre, ont contribué à la réalisation de cet ouvrage.

En particulier mes plus vifs remerciements vont aux membres du Jury d'Examen :

- à Monsieur le Professeur Max DAUCHET, qui me fait l'honneur de présider ce Jury

- à Monsieur le Professeur Vincent CORDONNIER, qui a assumé la direction de ces recherches ; ses encouragements, les discussions nombreuses que nous avons eues ont été le moteur de ce travail

- à Monsieur le Professeur Michel LUCAS, de l'Université de Nantes, qui, en plus de son amitié et de son humour, m'a apporté un soutien scientifique sans limite

- à Monsieur Francis MARTINEZ, maître-assistant à l'Université de Grenoble, dont les conseils et les idées, confrontées aux miennes, ont contribué à l'avancement de mes recherches

- à Monsieur le Professeur Claude GIRAULT, de l'Université de Paris VI, qui, jouant son rôle de "parrain" CNRS, m'a apporté de l'extérieur une vision neuve sur certains problèmes

- à Monsieur Gérard GAILLAT, ingénieur en chef à la Société MATRA, qui a montré un intérêt certain pour mes travaux et accepté de les juger

- à Madame Alice RECOQUE, conseillère scientifique à la Société BULL, et membre du Comité de la Section 08 du Centre National de la Recherche Scientifique, qui a également accepté de juger cette thèse.

Je tiens à remercier très chaleureusement tous mes collègues du Service Informatique et du Laboratoire Associé 369 pour les échanges de vues féconds que nous avons eu, et en particulier Madame Patricia CARON qui a assuré avec gentillesse et efficacité la mise en forme et la dactylographie de ce document, ainsi que Monsieur Henri GLANC qui en assuré le tirage.

Enfin il serait ingrat d'omettre ici tous les membres de ma famille, qui m'ont apporté la patience et la compréhension nécessaires à l'aboutissement de ce travail.

TABLE DES MATIÈRES

PRÉSENTATION

Ce travail est une contribution à l'"infographie" ou imagerie informatique. Il s'inscrit dans une démarche Architectures Spécialisées.

Il convenait tout d'abord d'effectuer quelques rappels au sujet de l'informatique des images ; il ne nous a pas paru nécessaire d'en faire un volumineux "pensum", car de nombreux ouvrages de références existent maintenant dans ce domaine. Ce premier chapitre se poursuit par l'étude générale des problèmes d'architectures adaptées à l'informatique des images ; nous y montrons qu'il est possible de définir trois grandes classes de machines répondant au problème, décrites dans les trois chapitres suivants.

Le chapitre deuxième décrit des machines-algorithmes existantes ; puis nous proposons les éléments de construction d'une de ces machines, à savoir des découpeurs bi-dimensionnels. Ce premier exemple nous permet déjà d'introduire des notions essentielles : la nécessité d'une étude fine et complète des algorithmes de base et l'importance de la conception d'architectures autour de circuits intégrés spécialisés.

Le chapitre suivant décrit dans la classe des machines-pixels une architecture cellulaire applicable à la fois au traitement et à la synthèse d'images. Elle nous conforte dans la nécessité de disposer de structures régulières pour faciliter l'intégration.

Le chapitre suivant rappelle succinctement les résultats déjà obtenus par Durif ([DUR 83]) sur les machines-objets.

Enfin, on trouvera en Annexe, d'une part une étude sur la recherche du parallèlisme dans un groupe de processus dépendants ou non, d'autre part une étude sur le sous-échantillonage qui permet de justifier certaines simplifications d'usage courant.

Il résulte de ce travail trois constatations

* l'étude des algorithmes de base est essentielle
* les composants à haut degré d'intégration et à très grande vitesse permettent seuls d'envisager la naissance de telles machines
* il y a convergence inéluctable entre synthèse et traitement d'images, y compris au niveau des architectures.

I - GÉNÉRALITÉS ET INTRODUCTION À QUELQUES ARCHITECTURES SPÉCIALISÉES

I - GÉNÉRALITÉS ET INTRODUCTION A QUELQUES ARCHITECTURES SPÉCIALISÉES

Il ne nous a pas paru opportun de rappeler ici toutes les notions de base de l'informatique des images : d'autres l'ont déjà fait, que l'on trouvera en bibliographie. Seules quelques notions que nous avons jugées les plus importantes sont développées rapidement, de manière à introduire la première partie de ce travail, c'est à dire l'étude d'architectures spécialisées pour les images.

I.1 - GENERALITES

I.1.1 - IMAGE

I.1.1 - *Définition de base*

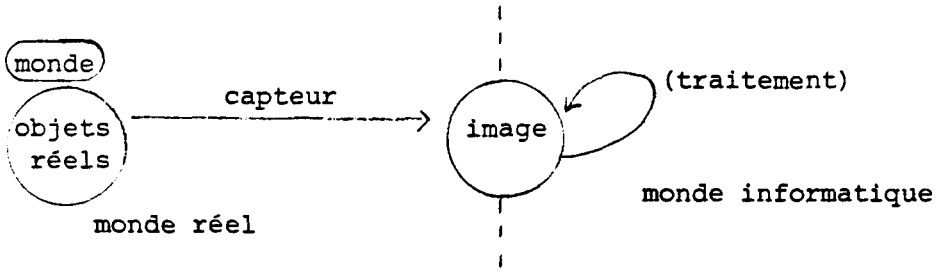
Le dictionnaire donne de ce mot de nombreuses définitions, dont nous retiendrons les suivantes :

a : représentation d'un objet par les arts graphiques ou plastiques

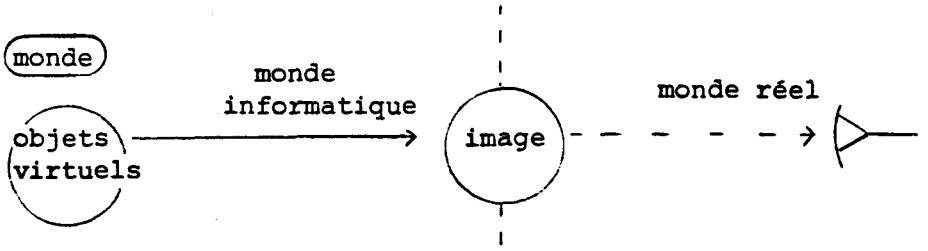
b : représentation mentale d'une perception (visuelle)

On note une confusion entre l'image physique et la représentation mentale que l'on s'en fait par la vue ; les problèmes liés à la perception seront étudiées dans la suite, mais nous nous attachons essentiellement ici aux images physiques, i.e. cherchant à représenter des objets par un procédé informatique.

Si les objets sont du monde réel, et si l'image est produite par un capteur à partir de ces objets, dans un but de traitement, d'analyse, de reconnaissance de forme, on a la chaîne



Si les objets sont virtuels, produits de l'imagination ou d'un programme, et si l'on cherche à en donner une image pouvant être appréhendée par divers observateurs, on obtient la chaîne



I.1.1.2 - *Autres définitions*

Divers termes sont proches de la notion d'image définie ci-dessus ; citons et définissons :

. dessin : représentation ou suggestion d'objets en se limitant à quelques informations significatives (lignes, traits, contour, etc ...)
. figure : représentation d'une forme par un dessin.

Nous nous refusons dans ce travail à cette simplification, pour chercher à être plus proche de l'aspect réel des objets. Une image sera donc perçue comme un ensemble de valeurs (gris ou couleurs) réparties sur une surface bidimensionnelle. La perception globale de ces valeurs donne l'image mentale associée.

I.1.1.3 - Modèles d'image

Il est difficile de donner un modèle conceptuel général d'images ; indiquons simplement deux modèles assez satisfaisants.

I.1.1.3.1 - Modèle de Bertin ([BER 77])

Ce modèle est applicable à des images bidimensionnelles constituées d'éléments séparables, que nous appellerons taches. On y définit huit "variables visuelles" permettant de définir une tache. Ce sont :
. la position (x, y) dans le plan (ce qui définit deux variables)
. la taille t (par rapport à une taille de référence)
. le grain (i.e. la texture)
. la couleur (i.e. teinte et saturation)
. l'orientation (par rapport à un centre de rotation et à une orientation de référence)
. la forme (allure du contour)
. la valeur (i.e. l'intensité)

I.1.1.3.2 - Modèle de Martinez ([MAR 82])

Il modifie et complète le modèle de Bertin : on y distingue
- . l'**I**dentification, qui permet de **nomm**er des **objets**
- . la **S**tructure, qui permet d'établir des relations (hiérarchiques) entre les objets
- . la **M**orphologie (forme)
- . la **G**éométrie (position, orientation, taille)
- . l'**A**spect (caractéristique intrinsèque de l'objet)
- . l'**E**clairement (caractéristique de l'environnement des objets).

Ce modèle est plus délibérément orienté vers l'algorithmique associée à la production d'images. On y voit apparaître clairement 3 classes d'informations, que les buts poursuivis par Bertin n'avaient pas permis de mettre en évidence. Nous y ajouterons, et cela rend à notre sens le modèle complet, la notion de temps, c'est à dire d'évolution de ces différentes informations au cours du temps. Cette dimension, orthogonale à toutes les autres, rend compte, de la manière la plus générale, de l'animation de l'image.
On notera que chacune des trois classes ainsi définie résulte du regroupement de deux types d'informations, d'une part des informations intrinsèques (identité, morphologie, aspect), d'autre part des informations extrinsèques (relations entre l'objet et le monde extérieur).

classe / type	logique	géométrique	matière
intrinsèque	identité	morphologie	aspect
extrinsèque	structure	géométrie	éclairement

I.1.1.3.3 - Les attributs

Chacun des attributs définis dans ces modèles peut être représenté ou modélisé de diverses manières.

Si l'identité et la structure sont des éléments généralement non visualisés sur l'écran, la morphologie et la géométrie peuvent être décrites de nombreuses manières : assemblage de formes ou de volumes élémentaires, description à l'aide de surfaces gauches, suites de points, etc ...

De même, aspect et éclairement se traduisent finalement sur l'écran par une certaine couleur, souvent décrite en termes de Teinte, Intensité, Saturation, et une certaine texture.

Mais la synthèse d'images réalistes met très souvent en oeuvre un modèle de calcul très lourd, basé sur les lois physiques de l'optique et la modèlisation fine du monde réel, de manière à restituer ombres portées, réflections, réfractions, transparence, ombrages, etc ... Dans ce domaine, la complexité des processus mis en jeu est sans commune mesure avec le résultat obtenu : si la production d'une image 1024 × 1024 × 24 bits nécessite 5 mn de calculs sur un Cray-1 de 10 Mips, cela signigie que l'on a effectué

$$5 \times 60 \times 10^7 = 3 \times 10^9 \text{ opérations}$$

soit **3000 opérations par pixel** !!

Une machine temps réel devrait donc avoir une puissance de 120000 Mips pour supporter cette modélisation.

Rappelons ici simplement l'un des modèles le plus simple de cette catégorie ([FOV 82]) :

l'éclairement en un point est la superposition de

. $Ed = RI_d$, éclairement "diffus"

- R : réflectance en (x, y, z)
- I_d : intensité de lumière ambiante

. ΣE_p avec $E_p = E_r + E_s$, éclairement des sources ponctuelles

→ $E_r = R \times \max(0, \langle L.N \rangle) \times I_p$.

- R = réflectance
- L = vecteur venant de la source
- N = vecteur normal à l'objet
- I_p = intensité de la source

→ $E_s = W(i) \times \max(0, \langle S.O \rangle^{**} n) \times I_p$

- W(i) = courbe caractéristique du matériau
- S = direction spéculaire
- O = vecteur vers l'observateur
- n = caractéristique de la brillance du matériau

. $E_t = T.I_t$, transparence

- T = coefficient de transparence
- I_t = intensité à transmettre

D'autres méthodes, telles que le "lancer de rayons" ([BOU 84]) amènent à une complexité très élevée des calculs.

I.1.1.4 - Machine d'images

On cherche maintenant à interposer une machine de traitement dans la chaîne objets-image-observateur ; cette machine a deux impacts essentiels :

a) l'homme interagit sur la machine en lui donnant des ordres au vu des images que celle-ci lui propose

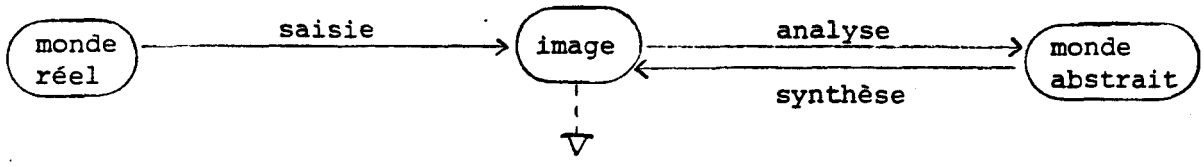
b) la machine, instrument numérique, impose de traiter des images dans un espace discret, donc non continu : ceci soulève tout le problème de l'échantillonage ("aliassage") et des conséquences néfastes qu'il

implique au niveau des traitements nécessaires à l'obtention d'images de qualité.
Nous pensons utile de définir trois classes de problèmes suivant leur schéma de relations entre le monde réel, l'image, et le monde virtuel (abstrait).

I.1.2 - TRAITEMENT D'IMAGES

I.1.2.1 - Schéma

Le schéma du traitement d'images (cette expression étant volontairement restreinte aux applications d'analyse d'images et de reconnaissances de formes) est le suivant



Les processus mis en jeu visent, à partir d'une image du monde réel

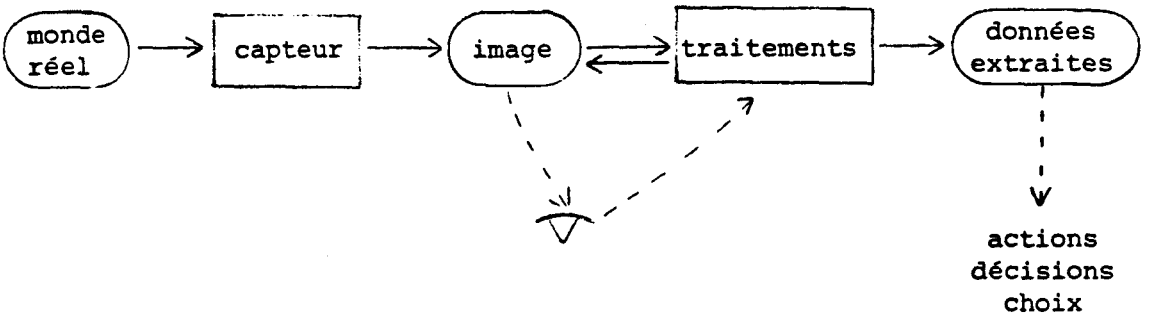
- soit à produire une nouvelle image après analyse et extraction d'informations
- soit à produire une structure de données représentant tout ou partie de ce monde réel.

I.1.2.2 - Algorithmes

Nous pouvons ici distinguer les algorithmes globaux, faisant intervenir les informations de toute l'image, et les algorithmes locaux, ne prenant en compte que des informations relatives à une zone réduite de l'image (un seul pixel, un pixel et ses voisins immédiats, etc ...). Citons dans la première catégorie : transformée de Fourier, histogramme ; et dans la seconde : suivi de contours, lissage. Un certain nombre d'entre eux seront détaillés ultérieurement.

I.1.2.3 - Machine associée

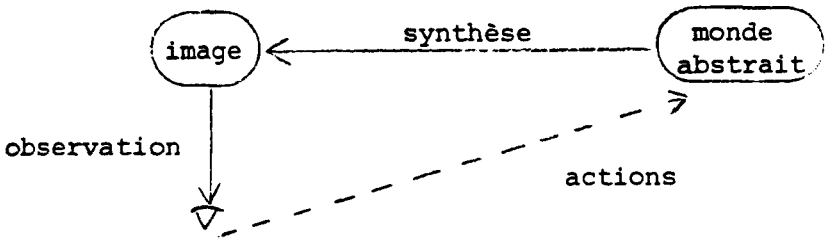
Dans les démarches classiques du traitement d'images, l'usager interagit peu avec la machine ; celle-ci a généralement une séquence de traitements bien définie. Parfois l'observateur est inexistant (robotique par exemple).
La machine a la structure générale suivante



I.1.3 - Synthèse d'images

I.1.3.1 - Schéma

Le schéma général est le suivant



Les processus mis en jeu visent, à partir d'une description abstraite d'un monde et/ou des actions de l'observateur à produire une image.

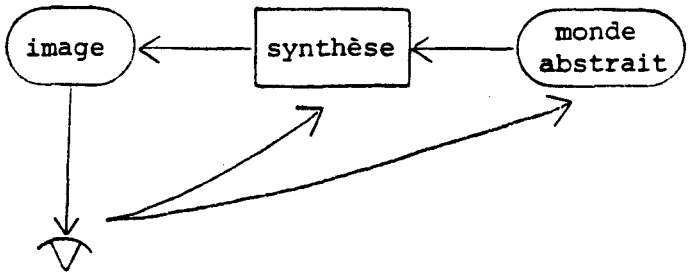
I.1.3.2 - Algorithmes

Une fois le monde abstrait défini les algorithmes consistent à interpréter ces données pour produire l'image ; ce sont par exemple :

- . les tracés de formes élémentaires (segments, cercles, etc ...)
- . l'élimination des parties cachées (cas d'un monde abstrait 3D)
- . l'ombrage, les calculs de transparence, de lissage, etc ...
- . la synthèse de couleur, de textures, etc ...

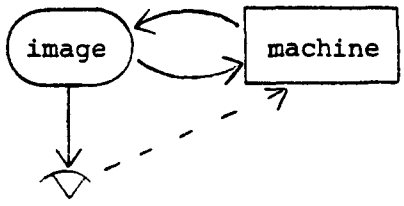
I.1.3.3 - Interaction

La construction complète ou en continu du monde abstrait peut relever de l'interaction de l'observateur avec la machine. Dans de nombreux cas, celle-ci est souhaitée et indispensable ; aussi doit-on mettre en jeu des moyens et des fonctions d'interactions assez élaborés pour répondre à ces besoins. On en déduit la structure de machine suivante :



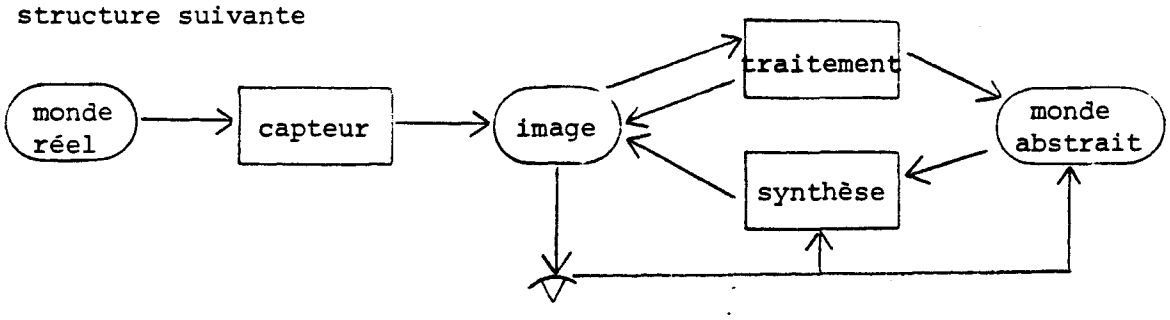
I.1.4 - MANIPULATION

Il s'agit simplement d'une réorganisation de l'image, ou des images, sans se préoccuper de sa sémantique ; donnons comme exemple les panoramiques, zooms, rotations, mais aussi l'incrustration, qui s'apparente plus à un traitement si elle est un peu élaborée. Au niveau structure de machine, on obtient



I.1.5 - CONCLUSIONS PROVISOIRES

* L'examen des graphes d'interaction ci-dessus met en évidence la dualité entre synthèse et traitement d'images. Une machine générale pourrait avoir la structure suivante



Cette dualité est plus profonde, car elle se vérifie du niveau des algorithmes et nous montrerons que certaines architectures peuvent satisfaire aux deux classes de problèmes simultanément.

* Dans tous les cas, il importe de fixer l'ordre de grandeur des entités manipulées : une image présentée sur un écran cathodique couleur actuel représente environ 300 K octets d'information ; une image cinéma 70 mm représente des dizaines de millions d'octets. Si une machine séquentielle effectue une opération chaque µseconde sur chaque octet, l'image sera totalement traitée en 0,3 à environ 10s, ce qui est très long. Une machine susceptible de traiter en temps réel une image 1024 × 1024 (soit un calcul complet tous les 1/25e de seconde) doit pouvoir traiter un point en

$$\tau = \frac{40\ ms}{1024 \times 1024} \simeq 40\ ns$$

Si un traitement représente 10 instructions, on obtient une puissance de 250 Mips, le débit moyen en sortie étant de 25 M mots/seconde !

On voit donc clairement l'importance d'une étude des processus mis en jeu et des architectures spécialisées adaptées à ce genre de problème.

* Une fois définis les concepts essentiels de ce vaste domaine, il semble nécessaire d'évoquer un certain nombre de problèmes et de voies de recherches que l'on trouve dans la littérature de manière assez disparate ([Luc], [New]) :

1) le dialogue homme-machine : analyse du comportement humain, formalisation du dialogue, dispositifs d'entrée/sortie, ...
2) les bases de données comportant des images (stockage, accès)
3) l'expression du mouvement, l'animation
4) la résolution du problème des lignes cachées
5) le sous-échantillonage
6) les textures, motifs, éléments génériques ...
7) l'obtention du temps réel vrai (1 image / 1/25e de seconde)
8) les algorithmes élémentaires
9) la compression d'image pour sa transmission et son stockage ; notion de complexité
10) les applications

etc ...

Un travail visant à déboucher sur des aspects concrets se trouve peu ou prou confronté à chacun de ces aspects. Or il ne saurait être question pour nous de les aborder tous.

Nous avons donc préféré centrer ce travail sur un thème privilégié. Il a trait à l'étude d'architectures spécialisées pour les images et inclut en particulier des aspects algorithmes élémentaires, complexité, temps réel, échantillonage, lignes cachées. La présentation en est faite dans la suite de ce chapitre et sert d'introduction à trois exemples d'architectures que nous considérons comme fondamentales et représentatives de trois catégories de base.

I.2 - INTRODUCTION A QUELQUES ARCHITECTURES SPECIALISEES

1.2.1 - ARCHITECTURES DE QUELQUES MACHINES EXISTANTES

1.2.1.1 - Monoprocesseur

De nombreuses machines courantes, en particulier les ordinateurs personnels disposant d'un "vrai graphique", sont dans cette catégorie ; la machine comporte, pour ce qui nous concerne ici

- un processeur exécutant les primitives graphiques
- une mémoire de trame accessible par ce processeur.

Deux configurations sont possibles :

1) la mémoire de trame est vue comme un périphérique et l'on y accède par l'intermédiaire de portes d'entrée/sortie
2) elle fait partie de l'espace d'adressage du processeur.

Eventuellement elle peut être accessible en Accès Direct Mémoire par blocs (dans le premier cas).

L'avantage de la solution 2 est évident ; supposons que le processeur a un temps de cycle constant de 1 µs. Si l'accès se fait par portes d'entrée-sortie, il faut :

- spécifier X et Y = 2 instructions d'affectation + 2 instructions de sortie
- lire ou écrire la donnée = 1 instruction

Total 5 instructions

Sans attente, le débit maximum est donc de $\frac{10^6}{5}$ = 0,2 M pixel/sec

Si la mémoire est dans l'espace d'adressage du processeur, ce débit devient 1 M pixels/sec (1 seule instruction).

L'expérience montre que l'accès par portes pénalise très fortement les performances. Son seul intérêt est d'être compatible avec n'importe quelle architecture non prévue initialement pour le graphique.

Notons cependant qu'une image 512×512 occupe 256 K mots ; il faut donc que le processeur accepte un tel "emprunt" sur son espace d'adressage.

A titre d'exemple, notre expérience sur un 8086 (5MHz) équipé de cartes Matrox donne

- accès I/O : environ 10 µs/pixel (1 image 512^2 en 3s)
- accès DMA : environ 0,8 µs/pixel (1 image en 1,2 s)

I.2.1.2 - Multiprocesseur + une mémoire de trame

Un certain nombre de machines sont construites autour d'un bus sur lequel sont connectés plusieurs (µ)processeurs, mais une seule mémoire de trame (en accès usuel, non partagé).
L'avantage est l'amélioration des performances de calcul mais les accès mémoire de trame sont inchangés ; le débit global n'est pas amélioré, car c'est la vitesse du bus qui limite les performances.

I.2.1.3 - Multiprocesseurs

Ils sont généralement conçus à partir d'un découpage géométrique de l'image (pavés, lignes ou colonnes) et d'une association statique processeur <-> bloc mémoire. Il en est ainsi de MPP (Goodyear Aerospace [BAT 80]) ou MAP ([RED 84]). Ces machines sont généralement assez puissantes et bien adaptées au traitement d'images. On observe généralement une forte influence de l'architecture sur les algorithmes et les performances : la zone mémoire élémentaire (pavé ...) tend à devenir l'élément optimal de calcul pour la machine.
On peut aussi classer dans cette catégorie les super-ordinateurs (CRAY, STAR ...) à usage général, utilisés fréquemment en synthèse ou traitement d'images. Ils sont généralement vectoriels, et ne sont pas forcément plus rapide qu'un monoprocesseur bien fait pour certains algorithmes ! De plus, ne disposant pas de sorties directes vers des visuels, ils ne peuvent travailler qu'en différé.

I.2.1.4 - Autres machines spécialisées

1. Le Générateur Synthétique d'Images ([LER 80])

Cette machine, destinée à faire partie de simulateurs de vol ou de conduite, réalise la visualisation en temps réel vrai d'images composées de facettes planes (1000 à 10000), ombrées, lissées. Elle est basée sur l'algorithme de SCHUMACKER ([SUS 74]), algorithme à priorités entre facettes.

La partie câblée temps réel de la machine comporte trois parties :

1) Prétraitement de l'image :
   - rotation et translation des objets
   - calcul de l'ombrage et préparation du lissage des facettes
   - évaluation des priorités dynamiques entre facettes

2) Chargement de l'image

Chaque facette est décomposée en trapèzes décrits par leurs bords droits et gauches, puis envoyée à la machine à tracer, dans l'ordre des priorités.

3) Machine à tracer

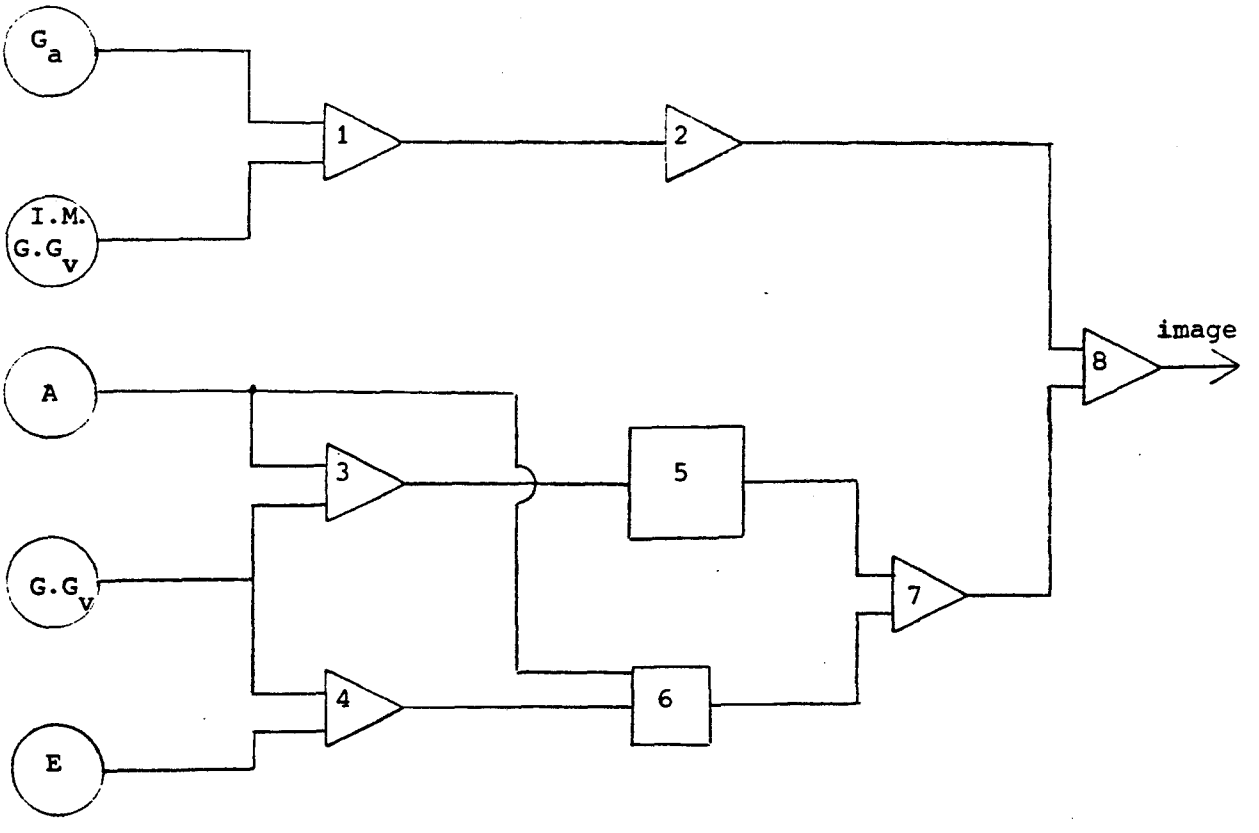
Elle trace dans une mémoire de trame chacun des trapèzes avec son ombrage effectif.

On retrouve le même genre d'approche dans d'autres réalisations orientées vers les simulateurs : peu importe le prix, on câble un ou plusieurs algorithmes connus (ici Schumacker et Gouraud), avec les moyens techniques nécessaires (µprocesseurs en tranches, logique rapide, ...).

2. La machine HELIOS ([MAR 82] et [FER 81])

Elle résulte de considérations bien plus fondamentales sur la nature des traitements mis en jeu dans la synthèse d'images réalistes : basée sur le modèle d'IMAGES décrit plus haut, c'est une architecture modulaire dont les éléments réalisent partiellement le processus de synthèse.

Le prototype réalisé effectue la synthèse d'une image suivant le schéma suivant :



Nota : G_v = paramètres de prise de vue

G_a = paramètres d'affichage

Les éléments de gauche sont élaborés par logiciel ; les opérateurs ou tables de droite ont les fonctions suivantes :

1 : fenêtrage de l'image, produisant l'I des faces visibles en chaque point
2 : la machine est multi-plans ; cet opérateur assure la comparaison entre ces plans
3 : projection de la texture sur la face, face par face
4 : évaluation des angles nécessaires au modèle d'éclairement
5 : table de textures
6 : table d'éclairement
7 : calcul effectif de A.E.
8 : synthèse finale.

A partir du modèle général IMAGES, d'autres architectures peuvent être déduites, se différenciant par les opérateurs implémentés. Cette approche originale est poursuivie, notamment vers la définition d'un "bus" de synthèse d'images, capable de supporter tout type d'opérateur.

I.2.1.5 - Conclusion

Les machines d'images existantes, à quelques exceptions près, ne sont pas des machines spécialisées : elles sont l'extension de machines classiques au problème des images. Celui-ci ne peut se satisfaire de cette approche car les informations à traiter sont trop nombreuses (plusieurs millions d'octets) et particulières (bi ou tri dimensionnelles) ; de plus l'exigence du temps réel (vrai ou lié à la capacité d'interaction) impose une approche parallèle.
Il est donc indispensable de développer des architectures spécialisées prenant en compte les spécificités de ce vaste domaine, et seules capables de satisfaire pleinement les exigences des utilisateurs.

I.2.2 - LES MACHINES A IMAGES

Dans ce chapitre sont définis un certain nombre de notions et de concepts qui apparaîtront souvent comme classiques et évidents ; néanmoins cette mise en lumière nous permet de proposer une classification méthodique, qui si elle est trop rigoureuse pour autoriser des "fantaisies" d'architectures, a l'avantage de fixer une référence, un cadre de choix et de comparaisons entre architectures.

1.2.2.1 - Principes

Une machine d'images est un système informatique limité d'un côté aux moyens d'actions (entrées) que l'utilisateur a à disposition, de l'autre à l'écran (aux visuels) que celui-ci peut regarder. Cette définition très large inclut toutes les machines actuelles produisant, traitant ou manipulant des images ou dessins.

Nous réduisons le cadre de l'étude en précisant que ce que l'on peut voir en sortie est une image bidimensionnelle, c'est à dire une surface assimilable à un tableau T(i, j) de valeurs visuelles (**pixels**).

L'application nécessite souvent la présence dans le système d'une représentation logique de l'image, que nous appellerons "scène", composée d'**objets**.

Les processus mis en jeu dans le système seront appelés "**actions**" (i.e. ce qu'il faut faire).

Ayant ainsi défini trois natures d'informations, pixels, actions et objets il en résulte de manière immédiate trois classes d'architectures suivant la préférence ou le centre d'intérêt que l'on a : une machine très proche des objets sera très différente d'une architecture implémentant des actions, etc ...

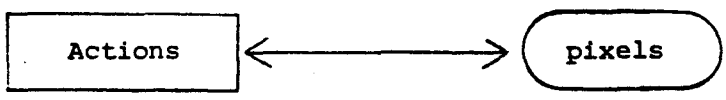
Une machine-objet est une machine organisée autour des objets de la scène, les actions à réaliser et les pixels de l'image finale ne devant pas cacher ces objets, mais contribuer à les faire apparaître ; une machine-action est une architecture cherchant à implémenter au mieux tous les algorithmes mis en jeu, par exemple en pipe-line ou sous forme d'un multiprocesseur MIMD*.

Une machine pixel, quant à elle, est entièrement orientée image finale ; l'exemple type est le tableau de processeurs, chaque processeur étant affecté à un ensemble de pixels.

1.2.2.2 - Domaines d'application

1. Manipulation d'images

Nous avons déjà indiqué qu'il s'agit là seulement de réorganiser une image, voire de réaliser des traitements simples entre images (inclusion, incrustation). Il n'y a donc pas d'objets à proprement parler et le schéma se réduit à

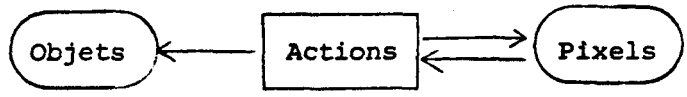


* *Multiple Instruction Multiple Data*

Comme on réalise généralement des actions simples et séquentielles, seule l'approche machine-pixel sera retenue. Nous trouverons deux limites : d'une part une machine où l'on associe une logique simple à chaque pixel, d'autre part une machine où cette logique est appliquée selon divers algorithmes de balayage à toute l'image.

2. Traitement d'images / Analyse d'images

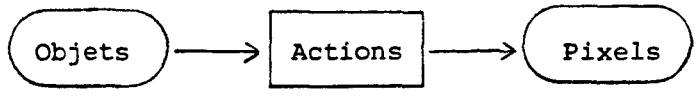
L'utilisateur souhaite réaliser sur des images un ensemble d'actions, pour obtenir soit une nouvelle image, soit un ensemble d'informations plus condensées, analogues à des objets. D'où le schéma suivant :



On voit clairement que deux des classes de machines sont possibles, d'une part les machines-pixels où les actions seront réparties sur chaque processeur-pixel, d'autre part les machines-actions, où chaque processeur fonctionnel effectuera l'ensemble du calcul associé sur tous les pixels.

3. Synthèse d'images

Elle met en jeu objets, actions, pixels, suivant le schéma suivant



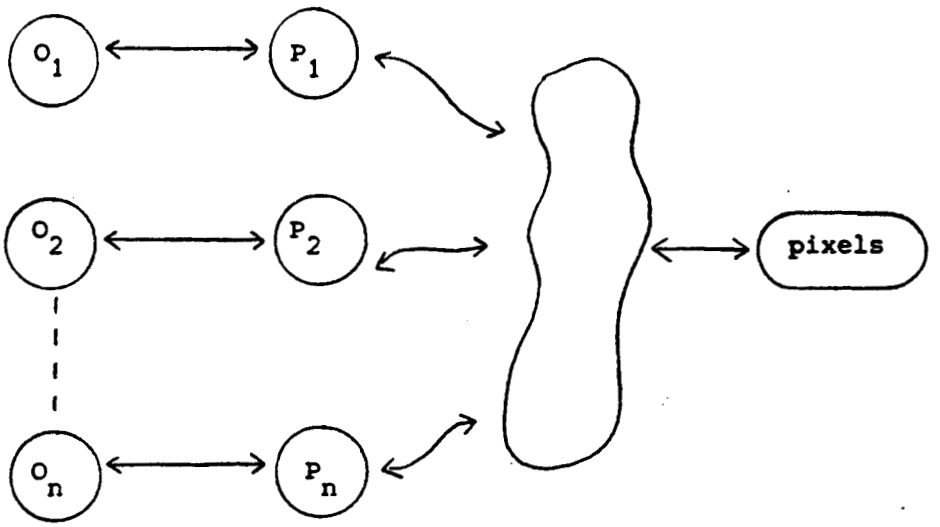
On pourra donc trouver ici les 3 classes de machines définies plus haut.

I.2.2.3 - Parallélisme

* Chacune des trois classes permet un certain degré de parallélisme, que nous allons préciser.

1. Machine-objet

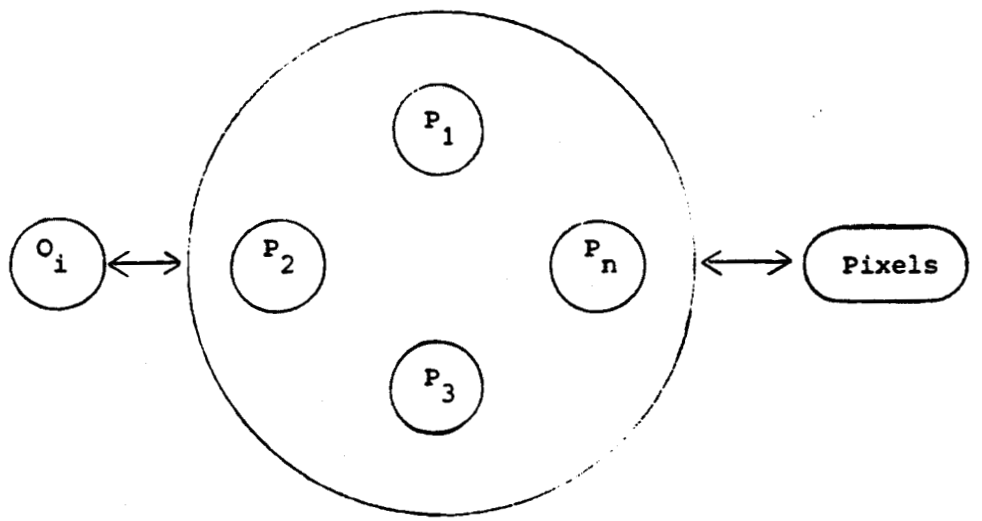
Une machine objet associe un processeur par objet de la scène, sur lequel il effectue tous les traitements. Si les objets sont indépendants, i.e. s'il l'on ne tient pas compte de relations entre objets, il est clair que le parallélisme potentiel est égal au nombre d'objets.



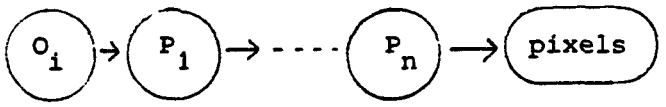
néanmoins, il reste à régler le problème d'accès à l'image (aux pixels). La machine proposée en II.3 montre une approche possible de traitement de ces conflits, à l'aide d'un bus intelligent.
Si les P_i réalisent les mêmes fonctions, on obtient donc une machine SIMD ou MIMD selon le degré de synchronisme entre les processeurs P_i.

2. Machine-Action

On associe un processeur à chaque action à réaliser, les objets étant traités séquentiellement. Le parallélisme est donc égal au nombre d'actions, si l'on est capable d'effectuer toutes les actions simultanément. Cela n'est généralement pas le cas, car l'ordre des actions n'est pas indifférent.

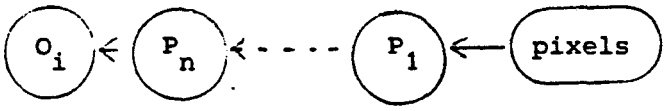


Un cas particulier très important est celui des machines pipe-line, soit



(synthèse)

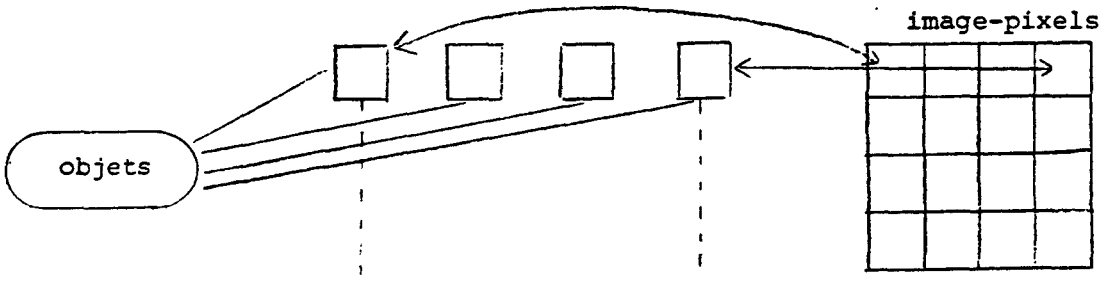
soit



(traitement et analyse)

3. Machine-Pixels

On associe un processeur à une zone géométrique de l'image, responsable du traitement de toutes les actions pour tous les objets dans cette zone.



Un cas particulier très important est celui des machines où l'on dispose d'un processeur par pixel ; une étude détaillée sera proposée dans la suite de ce travail (chapitre III).
Le parallélisme est ici géométrique et est égal au nombre de processeurs (de zones) mis en jeu. Nous montrerons qu'en réalité les processeurs ne sont pas indépendants et que certains ne sont pas concernés par tous les objets.

* De l'étude ci-dessus résulte le fait que le parallélisme est fonction de l'éventuelle dépendance entre éléments (processus). Nous supposons dans la suite qu'il existe un critère permettant d'affirmer si deux éléments sont dépendants ou non, i.e. si deux processus sont réalisables simultanément (ou en pipe-line).

1. Exemples de critères d'indépendance

. Dans un algorithme de lignes cachées, les objets peuvent généralement être partitionnés en sous-groupes, avec comme critère : deux objets de deux sous-groupes distincts peuvent être traités indépendamment. Nous pouvons donc créer autant de processus identiques que de sous groupes.

. Dans une machine-pixel, deux pavés disjoints peuvent effectuer des processus différents (voir chapitre III).
. Sur un même objet, certaines actions peuvent être parallèles, d'autres doivent être séquentielles (ou pipe-line).

2. Détermination du parallélisme optimal

Le parallélisme est maximum si l'on sait partitionner l'ensemble des processus (ou éléments) en un minimum de sous-groupes formés de processus (ou éléments) indépendants deux à deux.

On trouvera en Annexe une approche par la théorie des graphes de ce problème et une proposition de solution.

II.2.2.4 - Intégration à grande échelle

1. L'approche VLSI ([MEC 80], [LEW 83])

Les composants à très haut degré d'intégration, apparus depuis quelques années, nous imposent de repenser la conception des machines. En effet, s'ils offrent le potentiel pour de très hautes performances, en distribuant les calculs le plus possible, et donc en permettant d'atteindre un degré très élevé de parallélisme, ils exigent une philosophie de conception des algorithmes très différente : il faut absolument

- éviter la dichotomie processeur/mémoire
- augmenter l'homogénité et la régularité des composants et de leur interconnections (par une standardisation, par exemple)
- diminuer la complexité des communications (données et contrôle) et aussi la complexité arithmétique du calcul.

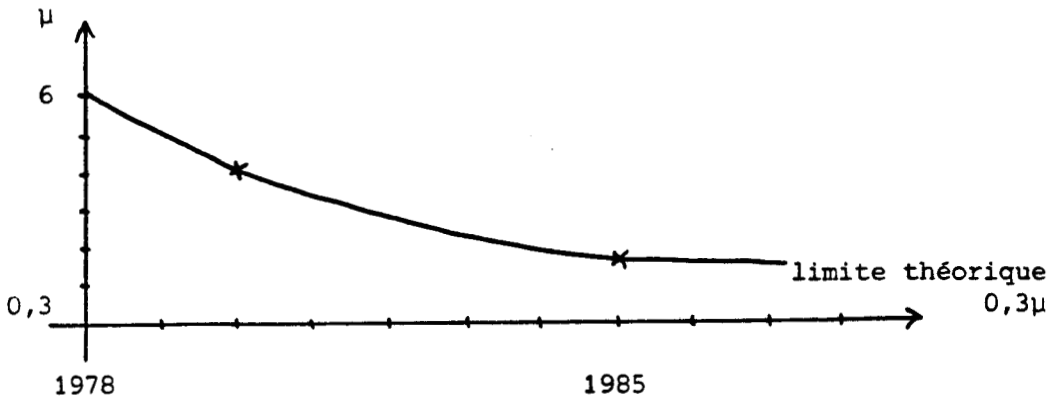
Une approche VLSI ne se justifie que compte tenu de ces remarques : il est inutile et illusoire de réaliser une machine comportant 1000 composants très complexes tous différents ; il est préférable de réaliser la même machine à l'aide de 5000 composants plus simples tous identiques, à la fois pour des raisons d'économie, de maintenance, de facilité de réalisation du logiciel associé, etc ...

D'où deux impératifs : simplifier les algorithmes à intégrer, et augmenter la régularité des composants, et par suite de ces algorithmes.

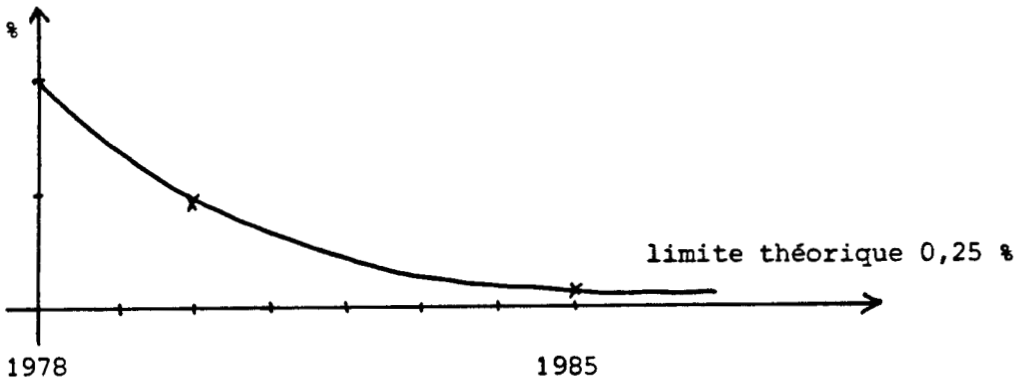
2. Les performances

Pour pouvoir discuter ultérieurement de la faisabilité de tel ou tel circuit, il convient de donner un ordre de grandeur de ce qui est réalisable.
Les trois graphiques suivants indiquent l'évolution de trois paramètres, la largeur des pistes, la surface active (100 % en 1978), et la fréquence d'horloge utilisable.

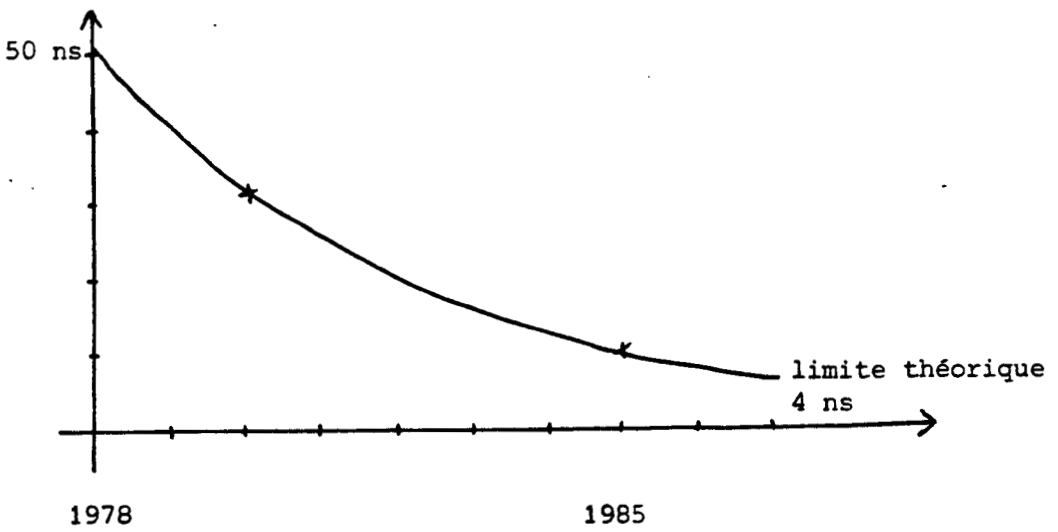
largeur de piste



surface active



horloge



Il est raisonnable de penser que les composants que l'on sait ou saura réaliser prochainement auront les caractéristiques suivantes :

. fréquence d'horloge > 100 MHz
. retard d'une porte < 0,5 ns
. densité > 20000 portes/chips
. nombre de connexions externes (pins) > 64

Prenons deux exemples de circuits de complexité moyenne :
- un additionneur 32 bits comporte environ 400 portes et induit un retard égal à 15 retards élémentaires ; on peut donc en réaliser plus de cinquante sur un boitier (en négligeant le problème des entrées-sorties !), qui exécuteront leurs additions en moins de 7 ns.
- un comparateur 32 bits comporte environ 200 portes et induit un retard de 5 retards élémentaires au maximum ; un boitier pourrait donc en contenir une centaine exécutant leurs comparaisons en 2 ns.

3. Applications à l'informatique graphique

Les utilisations spécifiques de composants VLSI sont actuellement assez limitées ; si l'on excepte les classiques générateurs de caractères alpha-numériques ou semi-graphiques, les contrôleurs de tubes cathodiques, on ne trouve guère que certains composants spécialisés pour les jeux vidéos, et quelques composants (NEC 7220 - EFCIS 9365) réalisant essentiellement du tracé de vecteur.

Très prochainement apparaîtront des extensions permettant d'engendrer des symboles graphiques (prédéfinis) en spécifiant forme, taille, orientation, emplacement, etc.

Mais nous verrons que l'informatique graphique soulève de nombreux problèmes susceptibles d'être résolus par intégration. En particulier, toute la partie temps réel de la synthèse d'image peut être et doit être pensée en termes de composants intégrés. Ainsi, Lewis ([LEW 83]) propose d'intégrer certaines étapes classiquement logicielles de la synthèse d'images : quatre applications sont proposées :

1) Un système de multiplication de matrices 4×4, basé sur quatre micro-processeurs, comportant 12 boîtiers et travaillant en entiers 16 bits (à rapprocher du "Geometry Engine" ([CLA 80]))

2) Une pile de tri, implémentant un tri sur 1024 éléments, destiné à réaliser les tris en x ou en z de la synthèse d'image.

3) Une pile de priorité, destinée à faciliter le traitement des surfaces visibles.

4) Un additionneur logarithmique, pour effectuer les interpolations dans les calculs d'éclairement.

Ces exemples montrent que la complexité des algorithmes, la quantité d'information manipulée (plusieurs millions d'octets), rend obligatoire de passer par les composants à haute intégration. Mais il est nécessaire d'étudier l'ensemble du problème de la synthèse ou du traitement d'images en fonction de son intégration possible et non pas de se limiter à tenter d'intégrer certains algorithmes les plus ardus ou les plus lents.

I.2.2.5 - Le problème de l'échantillonage

A la différence des machines à balayage cavalier ou des machines analogiques, les machines à mémoire de trame impliquent un échantillonage du signal au niveau valeur et au niveau spatial.

1) Dans le premier cas, il en résulte des "marches d'escalier" dans l'ensemble des valeurs possibles ; le remède est lié uniquement au nombre et à la disposition des valeurs affichables. Le cas le plus démonstratif est sans aucun doute celui de la "simulation de niveaux de gris", détaillé ci-après :

étant donné un support d'affichange à 2 couleurs, noir et blanc, ce support ne permet d'afficher directement que des "dégradés" simples noir/blanc ; si l'on souhaite produire des dégradés plus progressifs, on est contraint de simuler les niveaux intermédiaires (gris), au prix d'une réduction de résolution spatiale : on associe généralement à chaque niveau de gris souhaité une matrice $n \times n$ contenant une distribution de pixels noirs et blancs. Une telle matrice permet de simuler n^2+1 niveaux de gris différents.

Une méthode analogue est parfois utilisée pour les couleurs (Tektronix 4027 ou HP9845C par exemple). Indiquons simplement le résultat essentiel : le sous-échantillonage des valeurs ne peut être résolu qu'à l'aide d'un sur-échantillonage spatial. Encore faut-il prendre garde aux phénomènes visuels pouvant accompagner ce traitement (moiré, motifs périodiques ...).
D'autre part, en ce qui concerne plus spécifiquement les couleurs, les études menées tendent à montrer que l'oeil est sensible, c'est à dire sait distinguer, plusieurs dizaines de milliers, voire quelques millions de nuances. Il importe donc, si l'on ne veut pas obtenir d'effets désagréables d'escalier

de nuances (renforcés par l'effet de Mach), de bien prendre en compte cette donnée physiologique. De plus, la sensibilité de l'oeil n'est pas la même dans la Teinte, l'Intensité et la Saturation, et n'est pas linéaire pour chacun de ces paramètres : il conviendrait donc de disposer aussi de modèles de couleur tenant compte de cette sensibilité, qui seraient très loin du modèle Rouge, Vert, Bleu universellement utilisé. Enfin indiquons qu'il y a plusieurs interpolations possibles entre deux nuances, et que par suite la correction d'un sous-échantillonnage de couleur est difficile.

2) Dans le second cas, les phénomènes résultants sont de plusieurs ordres. Si l'on se reporte à la théorie, la fréquence d'échantillonnage doit être au moins deux fois plus grande que la fréquence maximale du signal (théorème de Shannon) ; sinon les fréquences élevées sont mal rendues et provoquent des signaux parasites (voir théorie de l'information et théorie du signal).

Dans une image, les fréquences élevées se produisent dans trois cas essentiels :

- . les petits objets
- . les bords francs d'objets
- . les zones à détails (textures)

La théorie n'indique qu'une bonne solution, le filtrage avant échantillonnage ; comme nous ne pouvons pas (en synthèse), assurer un filtrage parfait, à cause de la nature discontinue des calculs, une bonne méthode consiste à sur-échantilloner la scène à visualiser, puis à filtrer numériquement le résultat obtenu.

Notons cependant que d'autres méthodes se ramènent à cette solution, si l'on prend la précaution de les analyser d'un peu plus près :

- . petits objets : évaluation de leur contribution au pixel
- . zones à détails : maintien de données avec plusieurs niveaux de précision.

En ce qui concerne le traitement d'images, le problème est généralement résolu par le capteur, qui assure le filtrage et l'échantillonnage de la scène (réelle) observée.

CONCLUSION

A ce niveau, il ne nous appartient pas encore de tirer de conclusions qui seraient prématurées, notre souhait premier est un souhait de classification, c'est à dire de clarification et de cohérence. Les développements qui suivent quant à trois exemples vont nous permettre de justifier les trois points essentiels suivants :

1) La définition et l'étude précise des algorithmes élémentaires souvent négligés, est une condition nécessaire, quelle que soit l'architecture choisie ;

2) L'intégration est un moyen naturel d'implémenter des architectures de machines à images ; l'utilisation de composants tous identiques en est un aspect important.

3) Le problème général de l'échantillonnage peut toujours être traité, à la double condition que :

a) les algorithmes le prennent en compte, chacun à son niveau

b) l'architecture le supporte.

BIBLIOGRAPHIE

a) Ouvrages généraux et de sensibilisation

[BAR 74], [FOV 82], [FUR 76], [JO 80], [LUC 77], [MOL 76], [NES 79], [REC 83] , [PAV 82].

b) Machines, synthèse, traitement

[ACQ 82], [BAS 84], [BOU 84], [CLA 80], [DAL 81], [DAN 84], [DUR 82], [FUC 77], [FUP 81], [GRA 80], [HWA 83], [KAI 83], [KID 83], [KIL 81], [LER 78], [MAR 82], [MAR 84], [MER 81], [MER 84], [TAN 83], [WEI 81].

c) Images, réalisme, sous-échantillonnage

[ALL 84], [ANP 76], [BER 77], [BLN 76], [BLI 77a], [BLI 77b], [BRA 81], [BUI 75], [CAT 79], [CRO 77], [DOR 83], [MAR 82], [MER 82], [OLE 82].

d) VLSI

[BAS 83], [CLA 80], [FIK 83], [FRW 82], [KUN 82], [LEW 83], [MEC 80], [RIC 82].

II - MACHINE-ALGORITHME

INTRODUCTION

Nous développons dans ce chapitre la notion de machine algorithme, présentée au § I.2.2.1. Quelques exemples nous montreront dans une première partie que de telles machines existent et sont relativement usuelles. Une deuxième partie s'attachera cependant à montrer qu'il est nécessaire d'une part d'extraire le noyau central du ou des algorithmes visés, d'autre part, de bien étudier ce noyau, pour définir une machine optimale et efficace ; l'exemple choisi est celui d'un découpeur bi-dimensionnel utilisable dans plusieurs applications différentes. Nous verrons dans les deux parties que la conclusion de ces études est bien souvent la réalisation de composants nouveaux.

II.1 - QUELQUES MACHINES EXISTANTES

II.1.1 - LES COMPOSANTS SPECIALISES

Ces composants commencent à apparaître. Nous étudierons ici deux d'entre eux, le EFCIS 9365 et le NEC µPD 7220.

II.1.1.1 - Caractéristiques communes

Ces composants sont destinés à décharger le processeur principal du tracé des points dans la mémoire de rafraîchissement. La démarche conceptuelle vient de la constatation que, dans une machine à dessiner composée d'un processeur et d'une mémoire de rafraîchissement, il y a six niveaux d'activités.

1) L'élaboration des signaux de synchronisation et de contrôle de la mémoire de rafraîchissement, ainsi que l'exécution de fonctions liées à l'horloge (zoom, panoramique, gestion d'un photostyle).

2) La mise à jour de la mémoire.

3) Le calcul des pixels à mettre à jour (décodage d'un tracé à effectuer).

4) L'élaboration des paramètres nécessaires à ces calculs.

5) La gestion de la liste des tracés à effectuer.

6) La gestion des objets logiques vus par l'utilisateur.

Ces composants intègrent les niveaux 1 - 2 - 3 de cette description, et doivent être pilotés par un (µicro) processeur. Ils déchargent donc ce processeur de toute la gestion mémoire et écran et exécutent le calcul des pixels beaucoup plus rapidement (environ 100 fois plus vite que ne le ferait le processeur programmé.

Il est clair cependant que, si ces composants présentent un intérêt évident sur des machines "modestes" telles que les ordinateurs individuels, la démarche suivie est très classique et ne remet pas en cause la structure des mécanismes de production :

- base de données logicielle
- mémoire de trame câblée.

Les performances annoncées, de l'ordre du Méga pixel par seconde, montrent, compte tenu de la simplicité des algorithmes intégrés, qu'il n'est pas envisageable de les utiliser dans une machine temps réel susceptible de synthètiser des images réalistes : cela nécessiterait des performances plusieurs dizaines de fois supérieures.

II.1.1.2 - Le Graphic Display Processor EFCIS 9365 [EFCxx]

Interfaçable avec tout microprocesseur 4 ou 8 bits, il gère une mémoire de n plans de 64×64 à 512×512. Il permet de tracer uniquement des vecteurs (algorithme de Bresenham) avec 4 types de trait différents. Il est capable d'écrire 1,5 millions de pixels par seconde au maximum (vitesse de tracé instantanée), ce qui correspond à environ 900000 pixels par seconde en moyenne (en tenant compte du transfert d'informations de contrôle).

II.1.1.3 - Le Graphics Display Controller NEC µPD 7220 [NECxx]

Plus récent, donc plus complexe et plus performant, le GDC peut gérer une mémoire de 256 K mots de 16 bits, visualisés sous un format quelconque pouvant aller jusqu'à 1K×1K×4 bits. Il intègre le zoom (1 à 16), le panoramique et le défilement (indépendant) de deux zones.

Interfaçable avec tout microprocesseur, il dispose d'une pile de commandes (16 mots) et d'une mémoire RAM de paramètres autorisant un grand (?) parallélisme avec le processeur hôte. Il gère également un accès direct mémoire (en entrée et en sortie).

Il permet d'effectuer 6 types d'opérations différents :

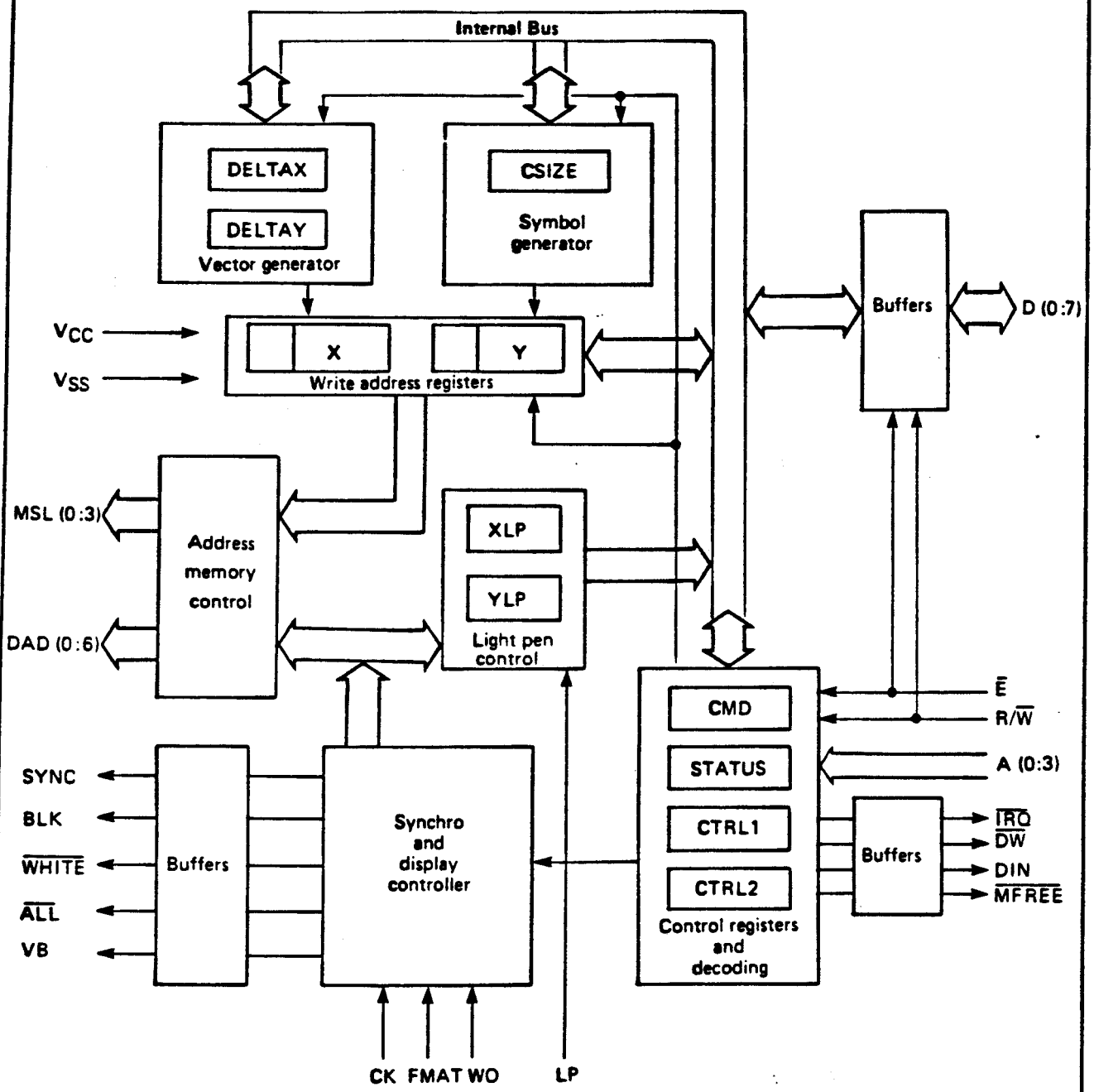
- tracé de segment
- tracé de cercle
- tracé d'arc de cercle
- tracé de rectangle
- tracé de caractère
- remplissage de zone (rectangulaire) par un motif.

Dans tous les cas, les performances annoncées sont de 1,25 méga pixels par seconde au maximum, ce qui doit correspondre effectivement à environ 800000 pixels/seconde en moyenne.

II.1.1.4 - Conclusion

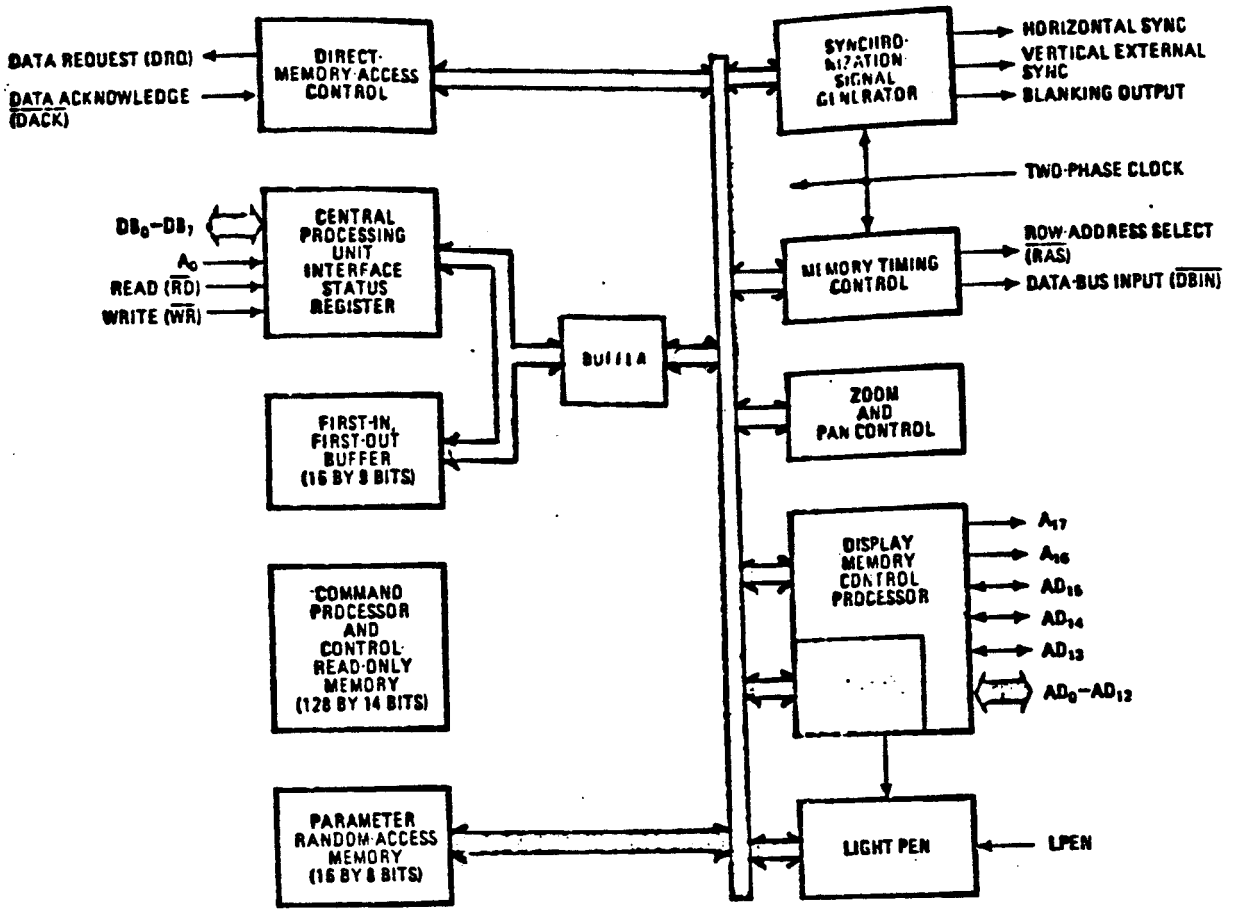
Même si la technologie évolue, par exemple d'un facteur 10 en vitesse, il semble que cette approche, très intéressante pour faire du dessin au trait sur des machines petites ou moyenne, ne pourra satisfaire les besoins de la synthèse d'images réalistes en temps réel vrai, c'est à dire la production d'une image toutes les 40 ms. Par contre, de tels composants sont utilisables pour du temps différé. De plus, beaucoup de problèmes ne sont pas pris en compte et devraient donc être traités au niveau logiciel, tels que ombrage, lignes cachées, sous-échantillonnage etc ...

BLOCK DIAGRAM



PIN DESCRIPTION

V_{SS}, V_{CC}	:	Power supply	
CK	:	Clock (TTL level ; 1.75 MHz for the CCIR standard : 625 lines 50 Hz	
FMAT, WO	:	Inputs for selectable functions	
SYNC, BLK, VB	:	Synchro and blanking signals	
DAD (0:6), MSL (0:3), $\overline{ALL}$	:	Display memory address signals	
$\overline{DIN}$	:	Data in	Display memory control signals
$\overline{DW}$	:	Display write	
$\overline{MFREE}$	:	Memory free	
D (0:7)	:	Data	MPU Bus
A (0:3)	:	Address	
R/$\overline{W}$, $\overline{E}$, $\overline{IRQ}$	:	Control signals	
$\overline{WHITE}$, LPCK	:	Light pen signals	



GDC : bloc-diagramme

II.1.2 - LE "GEOMETRY ENGINE" ([CLA 80], [CLA 82])

1.2.1.1 - Objectifs

Il s'agit de réaliser de manière cablée et parallèle (en l'occurence pipe-line) la plus grande partie du traitement géométrique nécessaire. Clark distingue et traite

- les transformations géométriques
- le découpage
- la mise en perspective.

Pour ce faire, est étudié et réalisé sous forme de VLSI un composant unique programmable.

II.1.2.2 - Le composant

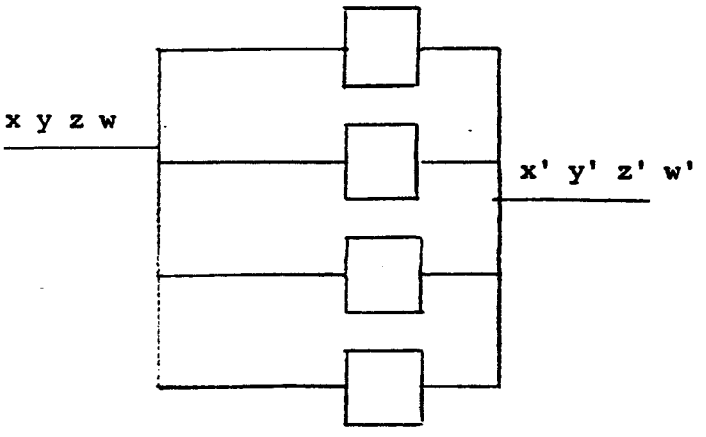
C'est une unité de calcul qui permet des opérations simples sur des nombres flottants (8 bits de caractéristique et 20 bits de mantisse). Elle réalise en parallèle sur les quatre composantes d'un segment en coordonnées homogènes des additions, des soustractions, et aussi (par décalage) des multiplications et des divisions. L'usager dispose d'un jeu de 12 opérations.

a) Les transformations géométriques

Elles sont réalisées à l'aide d'un produit matriciel du type :

$$[x\ y\ z'\ w'] = [x\ y\ z\ w]\ M$$

réalisé en parallèle par 4 composants, chacun effectuant un produit scalaire.



b) Le découpage

Réalisé en trois dimensions, il effectue les tests suivants

- $w' < x' < w'$
- $w' < y' < w'$
- $w' < z' < w'$

Basé sur l'algorithme de découpage réentrant de Sutherland-Hodjman [SUH 74], il nécessite 6 étages en cascade, chacun réalisant le découpage d'un segment par un plan de découpage.

c) La mise à l'échelle/perspective

Cette opération calcule non seulement le X et le Y que l'on obtiendra sur l'écran, mais aussi :

- Z, utilisable ultérieurement pour déterminer les lignes cachées
- D, valeur utilisable pour la génération de vues stéréoscopiques

Pour ce faire deux unités évaluent

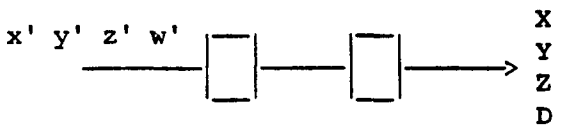
$$D = (z'/w') * S_s + C_s$$

$$Z = (z'/w') * S_z + C_z$$

$$X = (x'/w') * S_x + C_x$$

$$Y = (y'/w') * S_y + C_y$$

(C_i et S_i étant des paramètres de mise à l'échelle et de perspective).



Ainsi 12 composants réalisent l'ensemble des fonctions indiquées.

II.1.2.3 - Performances

Le composant de base est un circuit intégré à 40 pattes, comprenant 4 unités de calcul identique programmables. Le système complet, formé de 12 composants, a une puissance d'environ 4 M Flops. Le débit, en termes de segments traités est de l'ordre de 100000 segments par seconde. L'intégration totale des 12 composants devrait quadrupler la puissance du système.

Ce composant fait partie d'une station de travail graphique (IRIS de Silicon Graphics, Inc.) complète, qui, permet de traiter 65000 coordonnées par seconde. Complété par un système très performant de remplissage (44 M pixels/seconde !), cette station complexe (plusieurs 68010, 1 M octet de mémoire centrale, disque dur, interface Ethernet) montre l'intérêt de créer des composants spécialisés dédiés à certains algorithmes.

II.1.2.4 - Conclusions

Plusieurs remarques sont ici nécessaires :

. Il est indéniable que cette approche est très intéressante quant aux performances atteintes.

. Utiliser toute "l'artillerie" du calcul flottant et des produits matriciels pour réaliser finalement un affichage sur un espace discret de 512×512 ou 1024×1024 est caractéristique d'une démarche consistant à augmenter la puissance de traitement plutôt que d'essayer de réduire la complexité des algorithmes traités.
Ainsi en est-il de certaines réalisations programmées d'algorithmes d'images réalistes que l'on peut trouver dans les revues spécialisées.

. Cette machine ne traite que des polygones convexes. De plus elle ne traite pas la visualisation proprement dite, c'est à dire la génération de segments, le remplissage, l'ombrage, etc ...

II.1.2.5 - STYX [MER 79]

Ce prototype, développé à l'Université de Lille, peut être vu comme une machine implémentant l'algorithme du z-buffer dans un espace 2D 1/2 ; étant donné un "crible" contenant l'image binaire du contour d'une tache associée à un niveau de profondeur z_o donné, un processeur câblé assure l'inclusion de cette tache dans la mémoire de trame T au rythme du balayage de la manière suivante.

si $z(i, j) \geq z_o$ alors $T(i, j)$ inchangée
sinon $z(i, j) = z_o$
$T(i, j) \leftarrow$ tache traitée.

Cette machine déjà ancienne a été réalisée en logique câblée ; elle pourrait être intégrée dans un seul composant.
Son intérêt essentiel est la notion de tache de couleur implémentée dans le matériel, associée à la notion de profondeur. Par contre son inconvénient majeur est la lenteur relative du processus : il faut 40 mn pour "incruster une tache dans l'image".

II.1.3 - AUTRES MACHINES

Nous avons présenté au chapitre I d'autres machines que nous pouvons classer dans les machines algorithmes : ce sont en particulier le GSI de Leray et Helios de Martinez. Cette dernière, en particulier est à considérer comme

une machine-algorithme si tous les objets sont traités par toute la machine. S'il y a des modules spécifiques à certains objets, on pourrait la classer dans les machines-objets.

II.1.4 CONCLUSIONS (DE CET EXAMEN)

Trois conclusions s'imposent à ce niveau :

1) Il est très intéressant d'étudier et de réaliser des machines-algorithmes : le gain de performances peut aller de 10 à plus de 1000 pour un algorithme donné.

2) Si on spécialise un composant pour un algorithme de haut niveau, ce composant figé est inutilisable pour d'autres usages et risque d'être incompativles avec des composants analogues.

3) Si les VLSI autorisent maintenant de peu se préoccuper des simplifications de détail des algorithmes, il n'en reste pas moins vrai qu'un bon choix initial d'algorithmes est nécessaire : si un algorithme calculant en entiers est possible, pourquoi intégrer à tout prix une machine calculant en réels sur 80 bits, même si l'intégration est possible ? C'est là une démarche classique de mauvais programmeur : si un programme fonctionne, pourquoi chercher à l'améliorer !

II.2 - UNE MACHINE ALGORITHME PARTICULIERE : LE DECOUPEUR BIDIMENSIONNEL

Introduction :

De manière à pouvoir utiliser un composant pour plusieurs usages, nous cherchons à définir le squelette d'un algorithme assez général. Nous aurions pu par exemple choisir d'étudier un interpolateur linéaire (voir [DUR 83]) utilisable aussi bien pour tracer des segments, que pour effectuer un lissage ou un calcul d'ombrage. Nous avons préféré nous intéresser au découpage spatial, présent dans plusieurs problèmes que nous allons détailler.

II.2.1 - TROIS PROBLEMES DIFFERENTS EN APPARENCE

II.2.1.1 - Les surfaces visibles : l'algorithme de Warnock [BOU 80], [WAR 69])

A - Il existe une large gamme d'algorithmes de surfaces visibles ; ceux-ci ont été longuement étudiés et comparés ([SUS 74], [BOU 80]). Parmi eux, on trouve une classe d'algorithmes procédant par divisions successives de la scène, de manière à traiter des zones où le nombre et la complexité des éléments sont faibles. Dans cette classe on range la proposition de J.G. Griffiths ([GRI 75], celle de J. Encarnacao ([ENC 75]) et l'algorithme de Warnock que nous allons détailler. Cet algorithme est caractérisé par les points suivants :

- la scène analysée est composée de faces planes
- il peut produire des surfaces ou portions de surfaces visibles, qui sont des zones rectangulaires (ou carrées) résultant de subdivisions successives de l'écran
- il travaille dans l'espace écran, se limitant à la résolution de celui-ci.

Le principe de l'étude de visibilité est d'essayer de "comprendre" une portion de scène à l'intérieur d'une fenêtre. Si cette scène n'est pas comprise, on divise la fenêtre (mère) en 4 sous-fenêtres (filles), que l'on étudie alors. Ce processus récursif s'arrête ou bien lorsque l'on a tout compris, ou bien lorsque l'on atteint la résolution désirée. La fenêtre d'observation initiale est égale à l'écran.

B - Les différents éléments de l'algorithme

Ba - La scène traitée est constituée de faces polygonales planes, convexes ou non. En principe aucune face ne doit être trouée, mais les faces peuvent se pénétrer.

Bb - Il est souhaitable de prétrier cette scène en z de manière à favoriser le traitement ultérieur. On choisit généralement le critère de tri suivant : une face F_1 est dite devant une face F_2 si et seulement si le point le plus proche de F_1 est plus proche de l'observateur que le point le plus proche de F_2. Le tri utilisé est souvent le "Quick sort", tri en 0 (n log n) en moyenne et $0(n^2)$ au maximum.

Bc - Un certain nombre de primitives sont utilisées par l'algorithme

α Projection.

On utilise une projection orthogonale classique

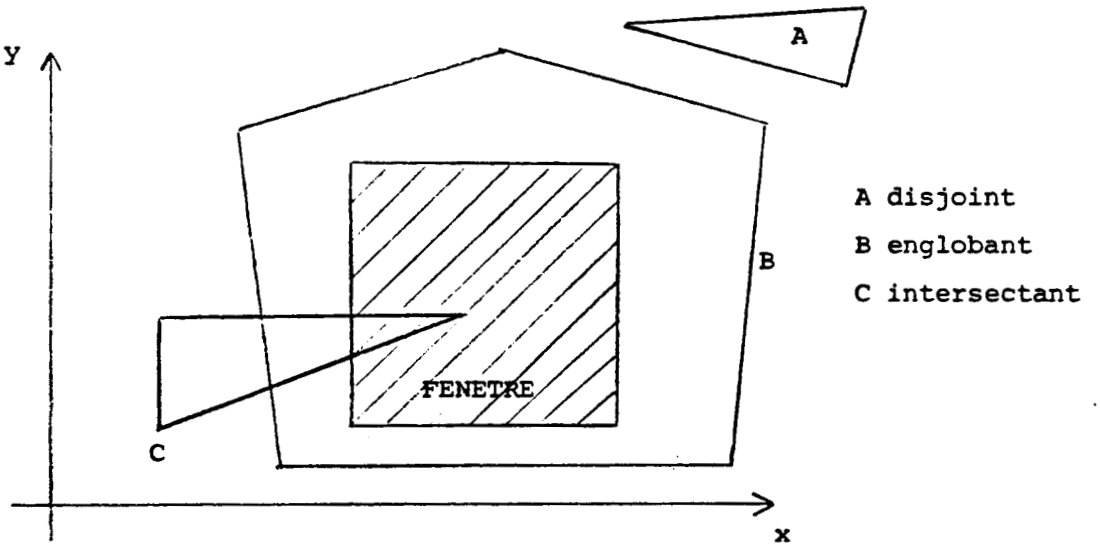
$$M(x, y, z) \rightarrow M'(x, y)$$

pour obtenir l'image 2D de la scène 3D.

β Relation de position

Cette fonction doit déterminer la position d'un polygone par rapport à la fenêtre d'étude. Trois cas sont possibles :

- intersectant
- disjoint
- englobant

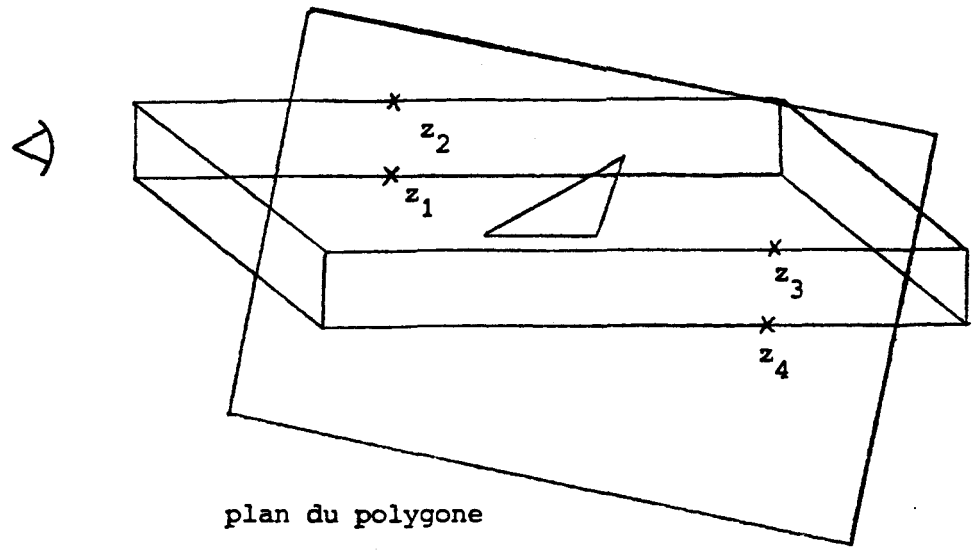


Relations de position

Nous verrons ultérieurement une solution partielle à ce problème.

γ Relation de profondeur

Il s'agit de déterminer, à l'intérieur d'une fenêtre, lequel de deux polygones est devant l'autre. On compare généralement les profondeurs des intersections de plan de chaque polygone avec les sommets de la fenêtre.



Relation de profondeur

Soient i et j indices de 2 facettes

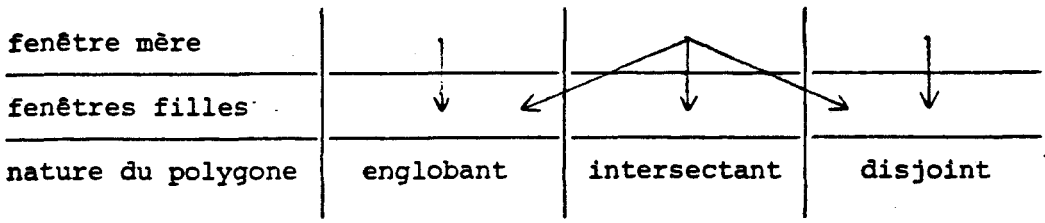
k indice de l'intersection ($1 \leq k \leq 4$)

1) Si $\forall$ k $z_{ik} \leq z_{jk}$ alors $F_i > F_j$

2) Si $\forall$ k $z_{ik} \geq z_{jk}$ alors $F_j > F_i$

3) Si $\exists$ k, k' $\begin{cases} z_{ik} < z_{jk} \\ z_{ik'} > z_{jk'} \end{cases}$, alors $\exists$ intersection des plans de facettes dans la fenêtre et il y a intersection possible des facettes.

Si l'on échoue dans l'étude de position et/ou de profondeur, on décide de subdiviser la fenêtre.
Dans ce cas ou obtient la hiérarchie suivante

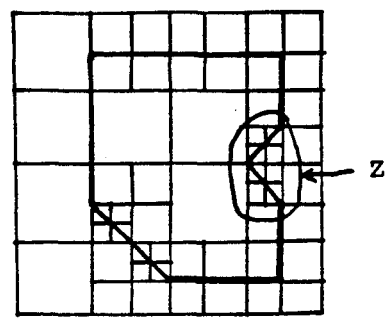


L'algorithme peut produire du dessin au trait ou une image.

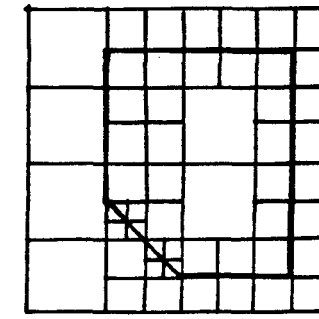
II.2.1.2 - Reconnaissance de formes par grammaire d'arbre

Une technique classique en reconnaissance de formes consiste à comparer un arbre extrait d'un objet à reconnaitre à des arbres de référence bâtis pour des objets connus. L'une des façons de construire cet arbre consiste

simplement à déterminer quel pixel ou groupe de pixels l'objet occupe. Bien entendu, pour que la reconnaissance soit possible, il faut pouvoir comparer deux arbres entre eux et par suite avoir un procédé de construction de ces arbres adéquat. Une solution consiste à affiner progressivement la description de l'objet ; ainsi une description par découpages spatiaux successifs peut être utilisée, moyennant des éventuelles (et souvent inévitables) translation et mise à l'échelle.



A

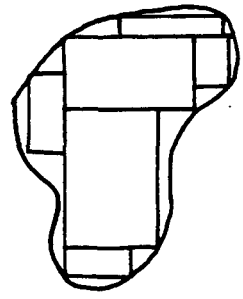


B

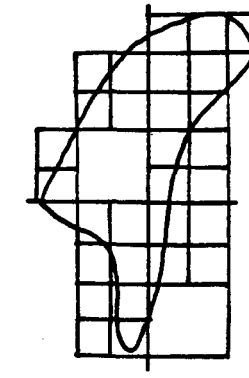
La différence entre A et B apparaît au moment du découpage de la zone encadrée Z.

II.2.1.3 - Remplissage

L'existence de mémoires inscriptibles par zones ([WHE 82]) permet d'améliorer notablement le temps de remplissage des contours. Ceci implique de savoir découper un objet en un ensemble (optimal si possible) de pavés rectangulaires ; si l'on peut imaginer une solution logicielle optimale, relevant des algorithmes de Recherche Opérationnelle, on voit immédiatement l'intérêt d'un algorithme plus simple, donc non optimal, basé sur un découpage aveugle de l'objet.



découpage (supposé) optimal



découpage aveugle

On obtient ainsi un arbre de présence à démonter pour effectuer le remplissage de l'objet.

II.2.2 - LE PRINCIPE

II.2.2.1 - Description

Les trois problèmes indiqués précédemment montrent à l'évidence la partie commune que l'on va chercher à implémenter. Il s'agit de déterminer, pour chaque objet, un arbre de présence dans l'image. Cet arbre est choisi quaternaire et résultera de l'analyse de l'objet par un dispositif de découpage récursif par des pavés carrés (ou rectangulaires).
Cet arbre pourra être produit

- soit en différé, avant l'exécution de l'algorithme principal
- soit en continu, pendant le déroulement même de l'algorithme principal.

Dans le premier cas, se pose le problème de la mémorisation des résultats alors que dans le second ce problème concerne les paramètres de récursion et les performances (vitesse).

II.2.2.2 - Utilisation

L'utilisation de ce noyau de calcul dans chacun des algorithmes étudiés est à peu près évidente :
a) En ce qui concerne l'algorithme de Warnock, il suffit de comparer les z des éléments en correspondance des arbres associés aux différents objets. Dans le cas d'une solution différée, se pose le problème de la recherche des éléments analogues dans des arbres différents, que nous n'aborderons pas ici.

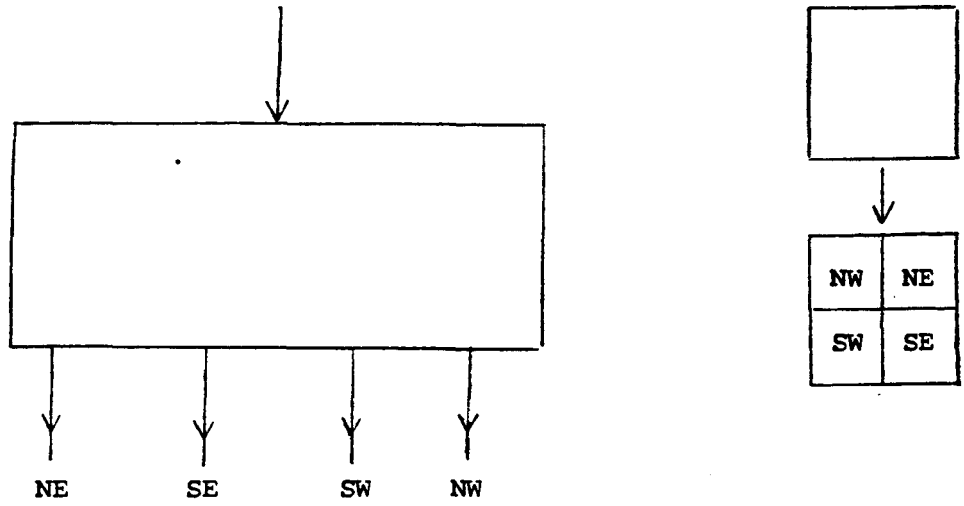
Dans le cas d'une solution directe, la construction des arbres se fait en paralléle pour les différents objets.

b) Quant à la reconnaissance de forme par grammaire d'arbres, le problème revient à comparer (à un niveau de précision donné) deux arbres quaternaires. Nous ne détaillerons pas cette question, qui trouve des solutions dans la théorie des arbres.

c) Le remplissage d'un objet dont on a l'arbre quaternaire consiste simplement à remplir les pavés feuilles de l'arbre à l'aide d'une valeur de marquage. Nous détaillerons ce processus ultérieurement, en annexe à ce chapitre

II.2.2.3 - Implémentation

Le dispositif utilisé pour fabriquer l'arbre est un découpeur spatial récursif. Nous proposons que la cellule de base soit un découpeur 2×2, dont l'entrée est associée à une fenêtre mère et les 4 sorties associées aux quatre fenêtres filles.



Etant donné un polygone P en entrée, une telle cellule fournit pour chaque sortie S :

$$\begin{cases} 0 & \text{si } P \cap S = \emptyset \\ 1 & \text{si } P \cap S = S, \text{ i.e. P englobe la fenêtre associée} \\ P' & \text{sinon, i.e. si P coupe strictement la fenêtre} \end{cases}$$

0 signifie que le pavé associé est vide ; il n'appartient pas à l'arbre

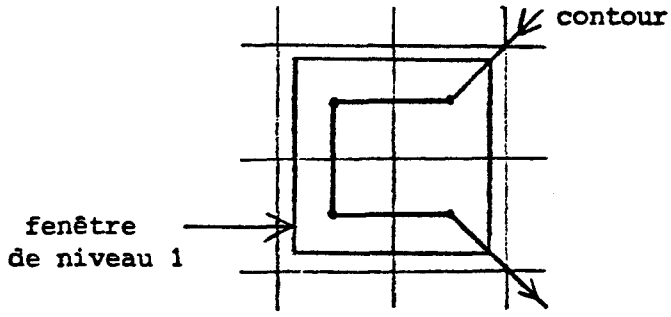
1 signifie qu'il est entièrement occupé par l'objet : on a obtenu une feuille de l'arbre

P' signifie que la récursion doit se poursuivre pour cette fenêtre. La fenêtre associée est un noeud de l'arbre.

Caractéristiques Générales

1) La récursion s'arrête au pire lorsque l'on a atteint la {résolution / précision} souhaitée R, qui correspond au pixel dans le cas de Warnock, ou bien à un sous-pixel dans le cas du traitement d'anti-aliassage par sur-résolution.
Ainsi, pour une image 1024 × 1024, il y a au plus 10 niveaux de récursion, l'arbre fourni ayant au plus 11 niveaux (y compris la racine (i.e. l'image toute entière) et le niveau pixel).

2) Les pixels constituant le contour d'un polygone sont produits par le niveau 0 (le plus bas) de l'arbre, sauf si le contour épouse une fenêtre de taille plus grande. Il en serait ainsi dans l'exemple suivant :



Si l'on considère que les directions de contour sont équiprobables, la probabilité de ce cas de figure est

$$4 \times 2 \times \frac{1}{7^3} \simeq 2{,}3\ \%$$

- $\frac{1}{7^3}$ → probabilité du parcours en  ou 
- 2 → sens de parcours
- 4 → points de départ

On s'autorisera par suite à négliger ces cas ; par suite, si P est le périmètre du polygone et n le nombre de descentes au niveau 0, on aura :

$$P \simeq n$$

II.2.3 - LA CELLULE ELEMENTAIRE : le découpeur unidimensionnel

Avant de détailler une solution parallèle et une solution mixte série parallèle, il est nécessaire de rappeler ici quelques résultats importants relatifs au découpage, au calcul d'intersection, etc ...

II.2.3.1 - Découpage d'un segment par une droite

Cette opération, essentielle dans notre étude, cherche à répondre à trois questions :

σ_1) une droite D coupe-t-elle un segment AB ?

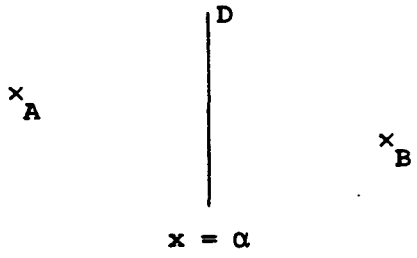
σ_2) de quel côté de D se trouve un point A (ou B) ?

σ_3) si D coupe AB, quelle est l'intersection I obtenue ?

Compte tenu de la nature particulière de la droite D dans les cas qui nous préoccupent (horizontale ou verticale), nous pouvons nous limiter à l'étude

d'une droite verticale d'équation $x = \alpha$, et d'un segment AB tel que $x_A \leq x_B$.
Dans ce cas, les réponses sont les suivantes

σ_1) Soit un segment AB connu par ses extrémité A et B et une droite D d'équation $x = \alpha$.
Nous dirons que D coupe AB si $\exists$ I | I $\in$]A B[et I $\in$ D.
En effet le cas I = A ou B est trivial et est traité par σ_2.

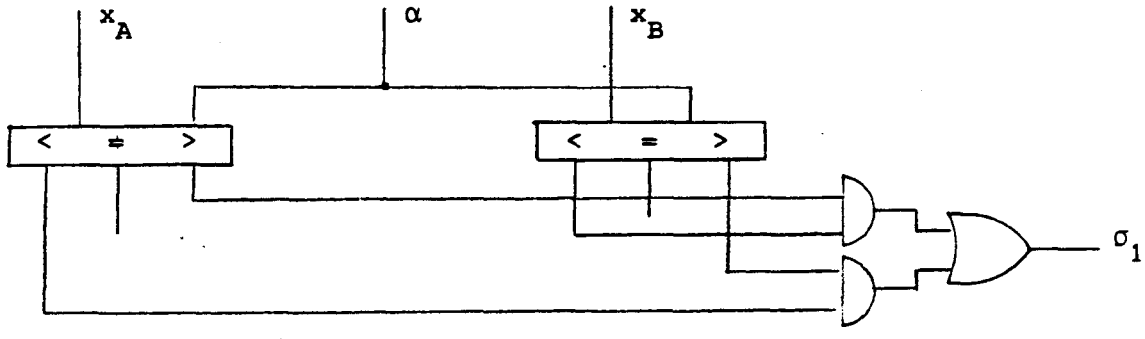


σ_1 (A, B, α)
si $(x_A - \alpha) * (x_B - \alpha) < 0$ alors renvoi vrai ;
sinon renvoi faux ;
finsi ;

N.B. La formulation mathématique ci-dessus peut se traduire plus simplement de manière câblée ; en effet, il n'y a pas lieu d'effectuer réellement les soustractions et multiplications, car seul le signe nous intéresse.
En réalité un algorithme plus logique s'écrirait :

$$\sigma_1(A, B, \alpha) = (x_A < \alpha \text{ et } x_B > \alpha) \text{ ou } (x_A > \alpha \text{ et } x_B < \alpha)$$

et s'appliquerait au circuit suivant.



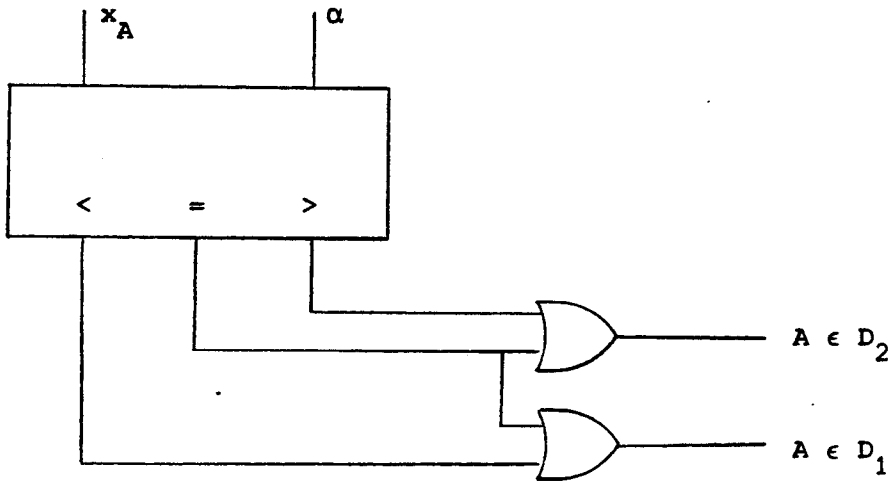
σ_2) Soit un sommet A et une droite D d'équation $x = \alpha$ délimitant deux régions $D_1(x \leq \alpha)$ et $D_2(x \geq \alpha)$.

L'algorithme et sa réalisation sont triviaux :

$\sigma_2(A, \alpha)$

si $x_A \leq \alpha$ alors $A \in D_1$
finsi

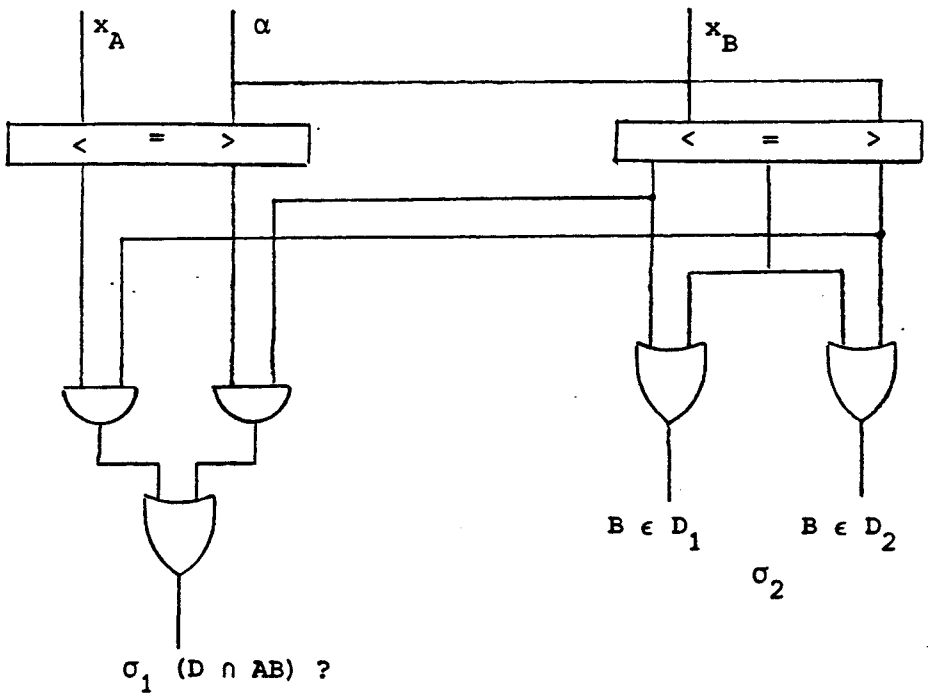
si $x_A \geq \alpha$ alors $A \in D_2$
finsi



Remarque :

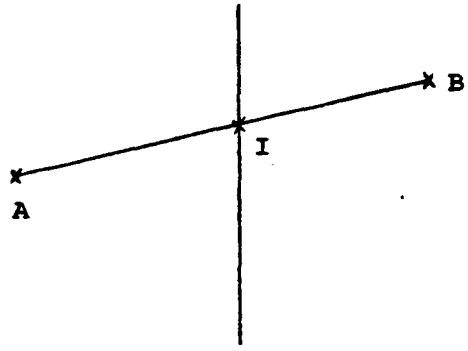
L'opération σ_1 fournit le moyen de faciliter la synthèse de σ_2.

D'où le schéma logique pour $\overline{\sigma_1}$ et $\overline{\sigma_2}$.



σ_3) Evaluation de l'intersection I

On suppose que AB coupe D en I ; il s'agit donc d'évaluer y_I.



1) Faisons l'hypothèse que $x_A < x_B$, $y_A \leq y_B$ ce qui ne nuit pas à la généralité de la démarche ; les cas $x_A = \alpha$ et $x_B = \alpha$ ont une solution immédiate et sont détectables par σ_1 et σ_2. Nous pouvons alors affirmer que

a) $y_A < y_I < y_B$

b) $y_I = y_A + \frac{\alpha - x_A}{x_B - x_A} * (y_B - y_A)$

est l'ordonnée exacte de l'intersection.

2) Cherchons à calculer plus efficacement l'ordonnée exacte y_I.

$$y_I = \frac{1}{x_B - x_A} [y_A\, x_B - x_A\, y_B + \underline{\alpha(y_B - y_A)}]$$

Seul le terme $\alpha(y_B - y_A)$ est à calculer, le reste ayant pu être pré-calculé. Le calcul se ramène donc (à peu près) à une multiplication et une division. Cette constatation montre que ce n'est sans doute pas la meilleure méthode de calcul, bien que l'on sache réaliser des multiplications et des divisions câblés (notamment par mémoires mortes).

3) Nous cherchons en réalité à déterminer non pas l'ordonnée exacte, mais l'ordonnée du point de D qui se situe le plus près du segment idéal AB (nous travaillons en entiers).

Une solution consiste à **tracer** le segment AB, à l'aide d'un algorithme efficace (Bresenham ou Loceff par exemple [DUR83] et à s'arrêter lorsque $x = \alpha$.

Ainsi l'algorithme de Loceff (en moyenne 8 × plus rapide que Bresenham) s'écrit : (si $\Delta y < \Delta x$) ([DUR 83]) ([LOC 80])

```
n = ENT (Δx/Δy)
e_n = n * Δy
y = y_A
tant que x < α
        y = y+1
        si e < Δx alors x = x+n ; e = e+e_n-Δx
                   sinon x = x+n+1 ; e = e+e_n+Δy-Δx
        finsi
fin
```

Il existe une version analogue pour $\Delta y > \Delta x$.
Il convient de plus de partir de l'extrémité la plus proche de α, pour gagner du temps.
Cette méthode ne convient pas à cause de sa longueur dans le cas de longs segments, ce qui sera le cas dans le début de l'algorithme de découpage (le nombre moyen de pas est $\frac{\alpha - x_A}{n}$).

4) Nous proposons donc une méthode par dichotomie :

Remarque

Une dichotomie directe avec évaluation d'erreur impose de tenir compte des parités des nombres manipulés. La méthode proposée ne gère ni erreur, ni parité.

Principe 1

1) Dans le cas où $\Delta y > \Delta x$, nous effectuons deux dichotomies identiques de précision suffisante sur y et x en même temps.

* La dichotomie suivante est déterminée par la position de x ⁄. α
* On s'arrête lorsque l'on a la précision voulue sur y.

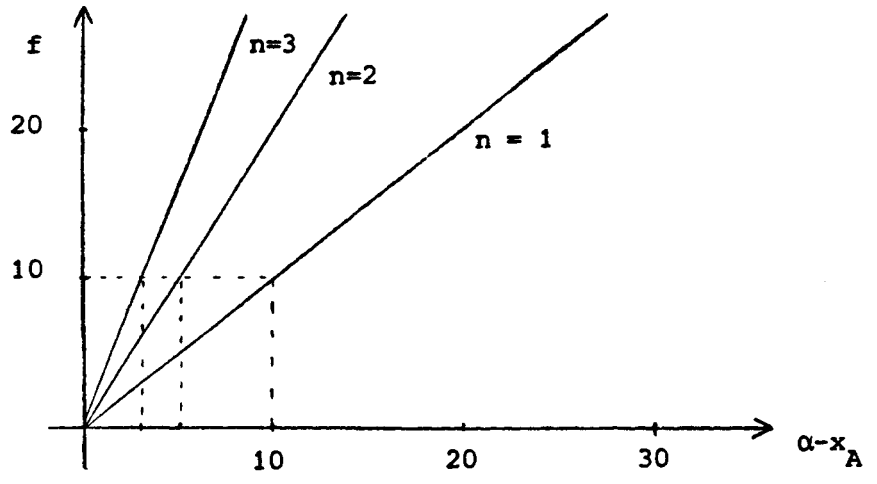
2) Dans le cas où $\Delta x > \Delta y$.

idem - On s'arrête lorsque l'on a la précision voulue sur x.

Il y a donc p pas au maximum si p est le nombre de bits de codage des informations.
D'où l'intérêt de l'algorithme si $p < \frac{\alpha - x_A}{n} = f$.

Si p = 10, il faut $\frac{\alpha - x_A}{n} > 10$ ou $\alpha - x_A > 10 \times n$.

→ exemple: longs segments de pente ≃ 1 (tels que A et B loins de α).

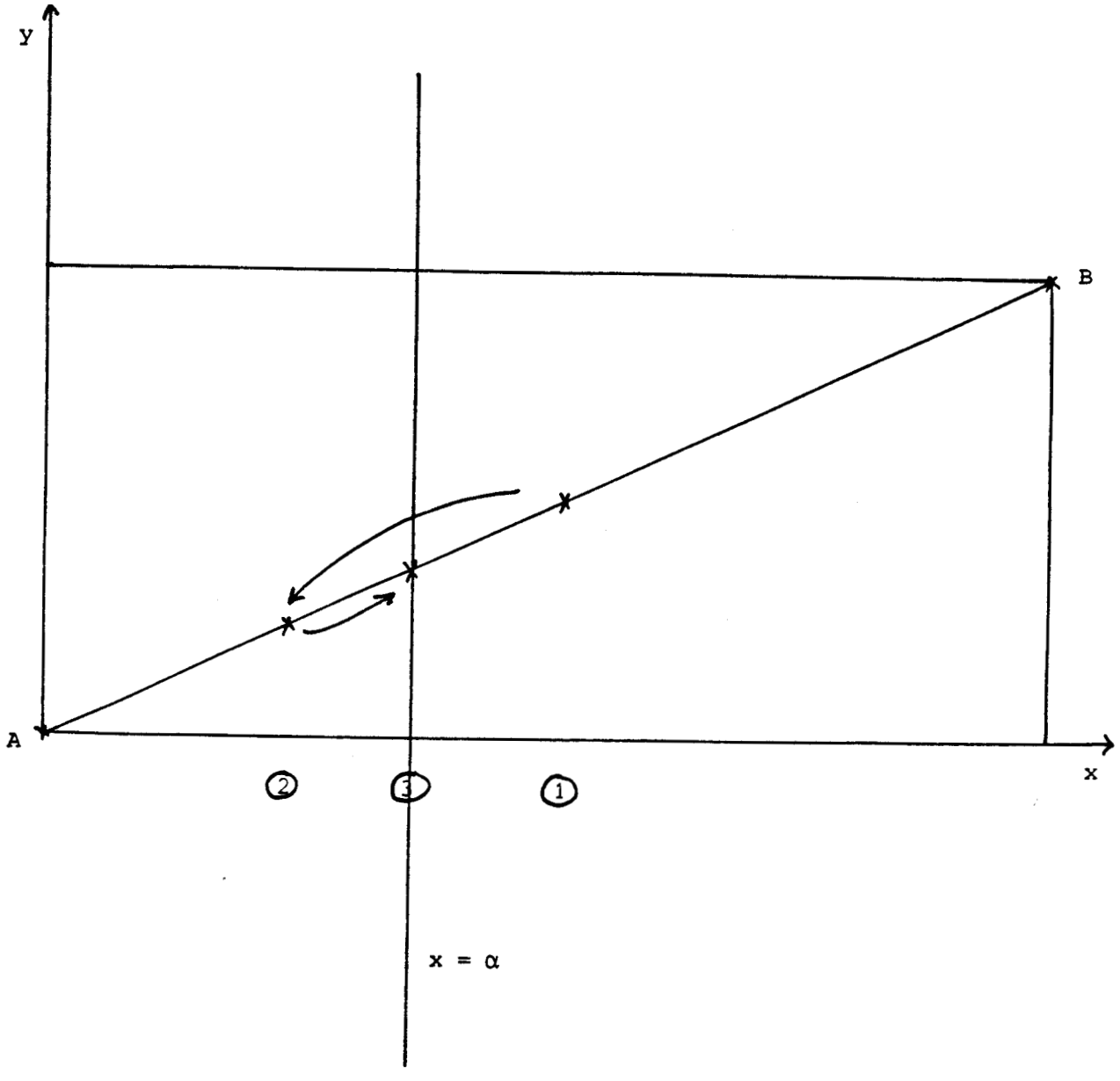


L'algorithme est (pour $\Delta y < \Delta x$).

$$y_1 = y_A + \frac{1}{2}$$
$$y_2 = y_B + \frac{1}{2}$$
$$\left.\begin{aligned} x_1 &= x_A - \alpha + \frac{1}{2} \\ x_2 &= x_B - \alpha + \frac{1}{2} \end{aligned}\right\} \text{ /* on ramène x à 0 pour faciliter le test de signe */}$$

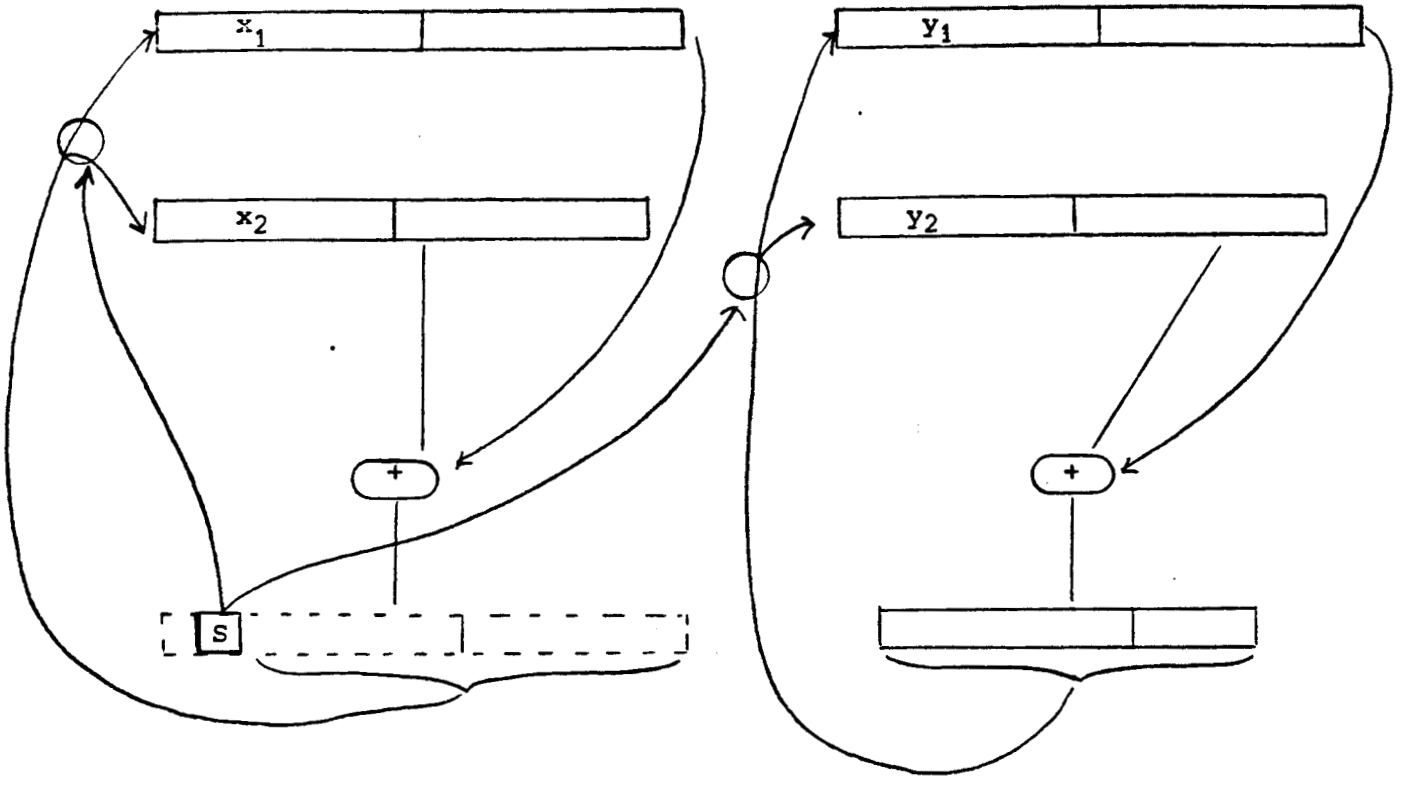
```
faire p fois /* au maximum */
   x = (x1+x2)/2 ; y = (y1+y2)/2
     cas  ENT (x)
     (<)  x1 = x ; y1 = y ;
     (=)  /* on a trouvé α */ retour  ENT (y)
     (>)  x2 = x ; y2 = y ;
     fincas
fait
```

Dans le cas où $\Delta y > \Delta x$, on effectue la boucle p fois et le test du **cas** se fait sur x ; y est obtenu en sortie de boucle



Exemple

Automate d'implémentation



* si p bits, alors les calculs se font sur 2p bits

* évaluation du temps de calcul pour 10 bits	1 + sur 20 bits	= 35 ns
	1 transfert registre	= 5 ns
	total par bit	= 40 ns

d'où t ≃ 400 ns (technologie TTLS, composants discrets).

5) Compte tenu de la complexité de ces algorithmes, nous proposons enfin une solution en PROM, valable pour les bas niveaux de découpage, ou ce qui est identique pour les faibles précisions.

En se ramenant à $x_A = y_A = 0$, on obtient une PROM à 3 entrées x_B, y_B, $\alpha(<x_B)$ et deux sorties x_I, y_I ; si n est le nombre de bits des entrées, le tableau suivant donne la taille de la ROM nécessaire :

n	ROM* mots	bits
2	32	4
3	256	6
4	2K	8
5	16K	10
6	128K	12

* ROM réellement occupée ; on tient compte de $\alpha < x_B$.

** pour n = 10, il faudrait une PROM de 512 M mots de 20 bits !!

n = 4 semble être une limite actuelle raisonnable ; ceci permet de déterminer immédiatement (t < 100 ns) l'intersection I dans un pavé 16×16.

6) En résumé nous avons 4 méthodes.

1) Calcul direct : 1 ×, 1/ (peu intéressant).
2) tracé du segment : intéressant si { pente forte (>> 1 ou << 1) ; segments courts
3) PROM : faisable sur zone peu étendue (faible précision)
4) dichotomie : durée constante intéressant sur longs segments

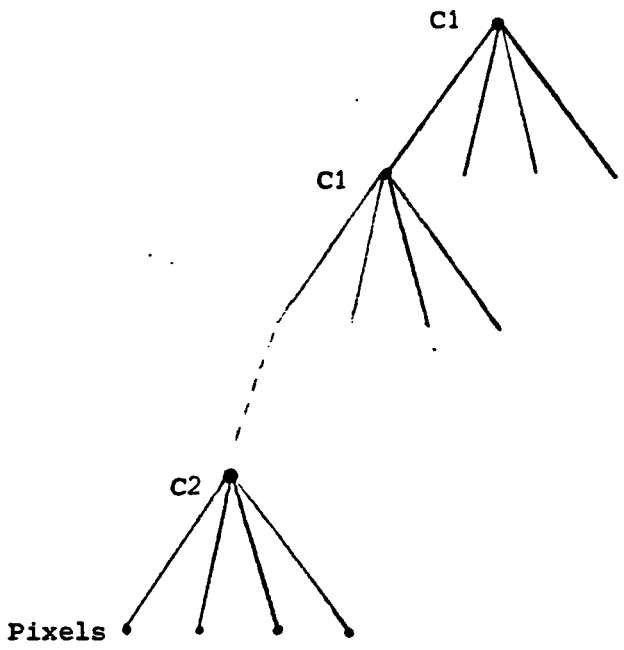
Des méthodes mixtes (3+4 par exemple) sont envisageables.

Nous proposons de retenir

C1 : la dichotomie dans les premiers niveaux de l'arbre

C2 : la PROM (ou le tracé de segment) dans les derniers niveaux (i.e. les 4 derniers niveaux).

Ceci signifie donc qu'il y aurait deux types de composants suivant le niveau de précision de leurs calculs.



II.2.3.2 - Découpage d'un polygone par une droite (d'après Sutherland/Hodgman [SUH 74])

1) Objectifs :

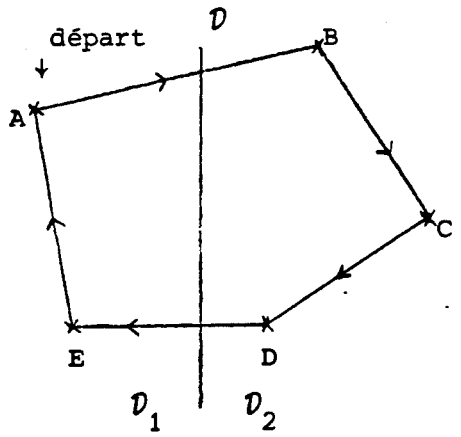
Cet algorithme fournit en résultat les polygones résultant du découpage par une droite D d'un polygone convexe fermé.

2) Hypothèses :

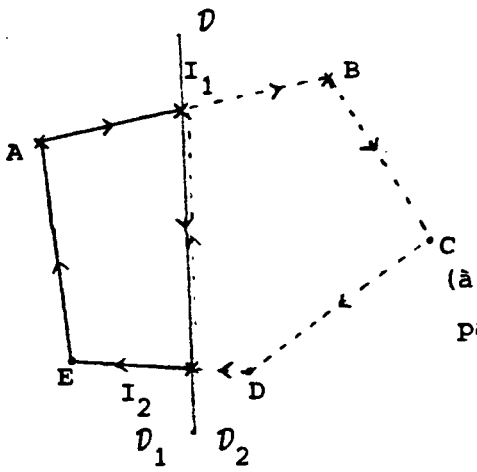
Une droite $\mathcal{D}$ divise le plan en 2 1/2 plans $\mathcal{D}_1$ et $\mathcal{D}_2$.
P sera le nouveau sommet à étudier, S l'ancien.

```
COUPER (P)
   si  (P non premier point et S P coupe D)
       alors  I = SP ∩ D ;
              I → D1 ;
              I → D2 ;
   finsi ;
P → S ;
si  S ∈ D1 alors S => D1 ; finsi ;
si  S ∈ D2 alors S => D2 ; finsi ;
fin
```

Exemple



polygone [A B C D E A]



résultat [A I_1 I_2 E A]

résultat [I_1 B C D I_2]
(à compléter pour fermeture par I_1)

3) Polygone quelconque :

Le traitement est possible de la manière suivante :

a) marquer les intersections I_j comme "entrant" ou "sortant" de la droite de division

b) Trier ces intersection le long de la droite de division

c) résoudre les segments résultants.

4) Evaluation :

Le traitement d'un sommet met en jeu les opérations suivantes :

σ_1 : un segment AB coupe-t-il une droite $\mathcal{D}$?

σ_2 : de quel côté de $\mathcal{D}$ un sommet se trouve ?

σ_3 : quelle est l'intersection I de AB avec $\mathcal{D}$ (si A coupe D) ?

que nous avons détaillées en 3.1. Retenons ici que le traitement d'un sommet représente

$$\begin{cases} \sigma_1 + \sigma_2 \\ \\ \sigma_1 + \sigma_3 + \sigma_2 \end{cases}$$ suivant le cas.

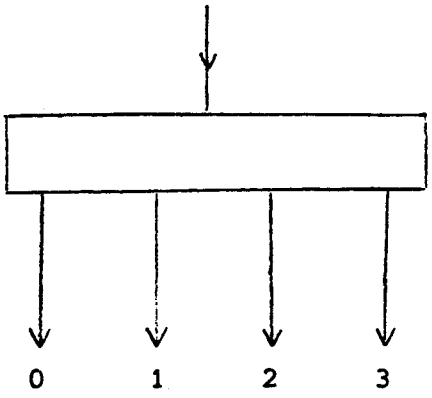
Notons que σ_2 et $(\sigma_1\ \sigma_3)$ peuvent s'effectuer en parallèle et que donc

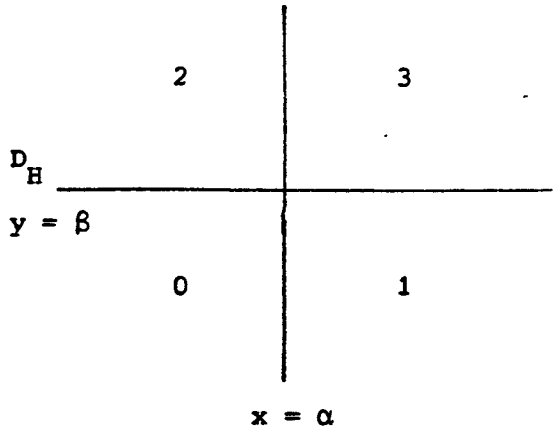
$$\bar{t} = t_{\sigma_1} + \frac{1}{2} t_{\sigma_3}$$

(en TTL/S, en choisissant une réalisation de σ par dichotomie, on aurait $t_{\sigma_1} << t_{\sigma_3}$ et $\bar{t} \simeq 200$ ns).

II.2.4 - LE DECOUPEUR BI-DIMENSIONNEL

Rappelons ici que nous cherchons à réaliser une cellule de découpage à une entrée (segment ou polygone) et à 4 sorties, correspondant aux 4 fenêtres filles dans un découpage bi-dimensionnel.



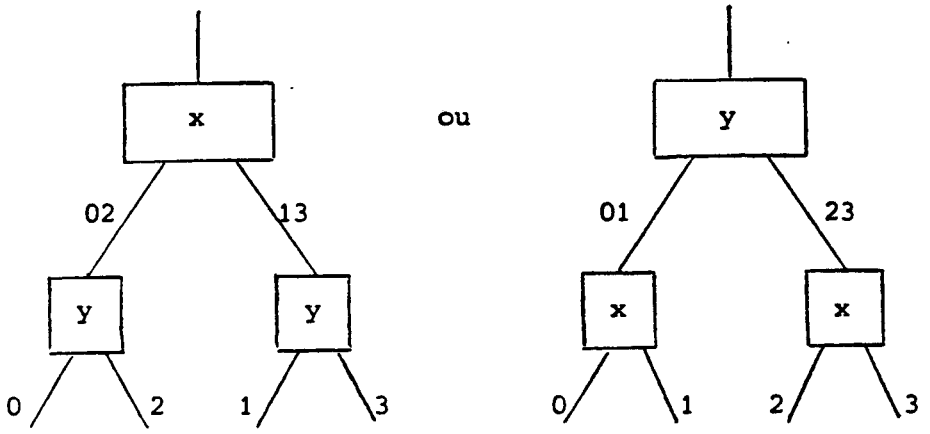


Nous examinerons successivement :

- deux propositions pour une cellule de découpage de segments
- le problème posé par le découpage de polygones
- les problèmes liés à l'algorithme de Warnock, i.e. la détermination du type de polygone produit (intersectant, englobant, ...)

II.2.4.1 - Solution séquentielle

Elle consiste bien évidemment à mettre en cascade deux découpeurs du type Sutherland-Hodgman, l'un en x, l'autre en y.
Il y a deux solutions strictement équivalentes :



Les modules de construction sont donc petits et tous identiques (un découpeur x ne diffère pas d'un découpeur y). Une cellule de base comporte 3 découpeurs ; si E est le temps moyen de traversée d'un découpeur, le temps moyen pour une cellule sera

$$\bar{T}_S = 2\bar{t} \qquad \text{(pour un segment)}$$

car il n'y a pas d'effet pipe-line pour le traitement d'un seul segment (les résultats d'un étage doivent être acquis avant la mise en route de l'étage suivant)

donc $\boxed{\bar{T}_S = 2t_{\sigma_1} + t_{\sigma_3}}$ ≈ 400 ns en TTL/S (avec σ_3 par dichotomie)

II.2.4.2 - Solution parallèle

1. Principes

Plutôt que de traiter séquentiellement les problèmes, nous cherchons à élaborer simultanément les quatre sorties. Pour ce faire, nous nous inspirons

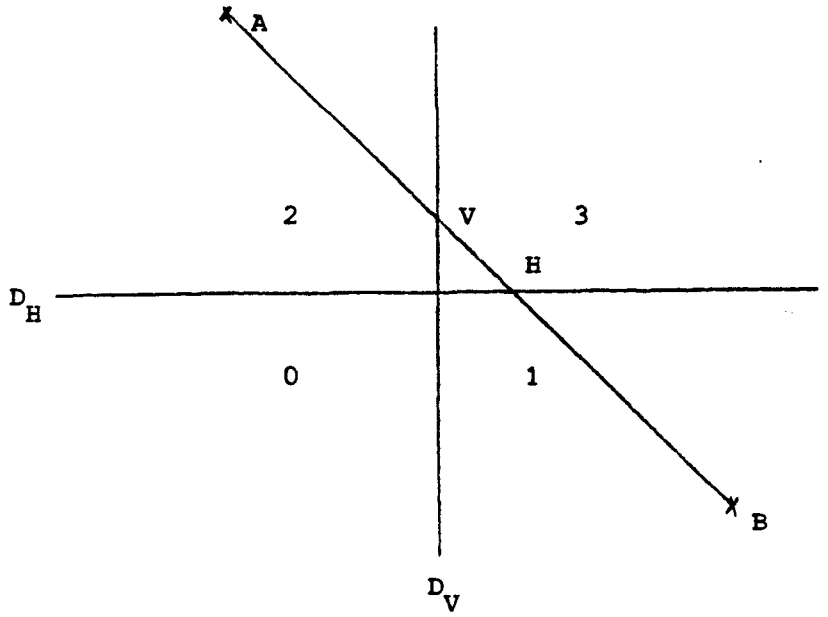
du découpeur élémentaire en l'étendant de la manière suivante ; aux 3 questions σ_1, σ_2 et σ_3 correspondant maintenant 3 nouvelles questions, qui sont :

Ω_1 : combien y-a-t-il d'intersections entre un segment AB et deux droites D_H et D_V ?

Ω_2 : dans quel quadrant se trouve un sommet A ?

Ω_3 : quelles sont les coordonnées du ou des points d'intersection (dans le cas où la réponse à Ω_1 n'est pas nulle).

Exemple :



0 ← ∅

1 ← HB

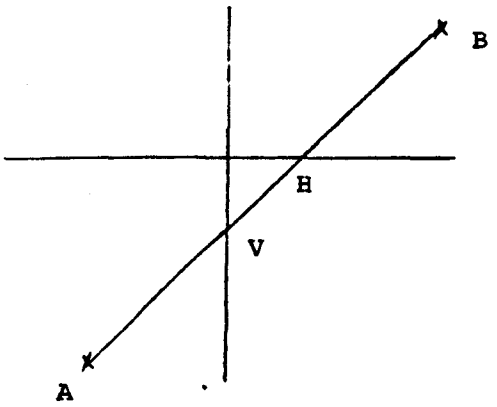
2 ← AV

3 ← VH

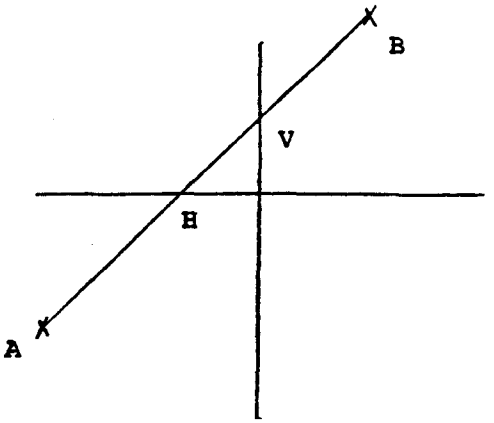
Cet exemple se généralise à l'aide du tableau suivant (où l'on suppose que A ∈ 0).

B	nombre d'intersections	intersections	segments résultants				
			0	1	2	3	
0	0	-	AB	∅	∅	∅	
1	1_V	$V = AB \cap D_V$	AV	VB	∅	∅	
2	1_H	$H = AB \cap D_H$	AV	∅	VB	∅	
3	2	$V = AB \cap D_V$	AV	VH	∅	HB	(1)
		$H = AB \cap D_H$	AH	-	HV	VB	(2)

avec la distinction (1)/(2) suivante



(1)



(2)

Cette distinction peut être obtenue de diverses façons, dont on retiendra

. l'évaluation du signe de $(\alpha-x_A)(y_B-y_A)-(x_B-x_A)(\beta-y_A)$

. la comparaison de y_V (ou x_H) avec β (ou α) **après** évaluation de V et H.

2. Implémentation

a) Ω_1 et Ω_2

Ω_2 s'obtient très aisément et Ω_1 s'en déduit immédiatement à l'aide de combinaisons logiques simples suivantes :

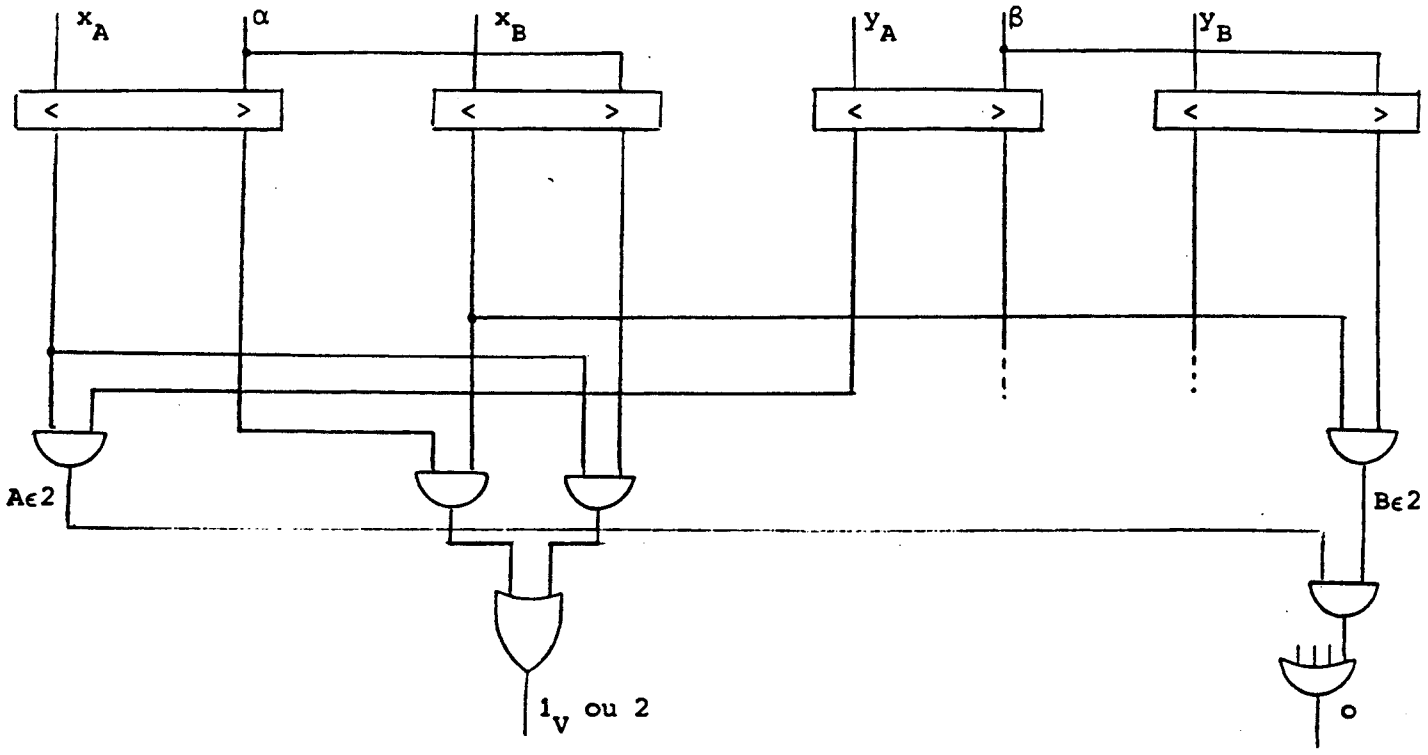
$$\begin{cases} 0 = (A \in 0 \text{ et } B \in 0) \text{ ou } (A \in 1 \text{ et } B \in 1) \text{ ou } (A \in 2 \text{ et } B \in 2) \text{ ou } (A \in 3 \text{ et } B \in 3) \\ 1_V = (A \in 0 \text{ et } B \in 1) \text{ ou } (A \in 1 \text{ et } B \in 0) \text{ ou } (A \in 2 \text{ et } B \in 3) \text{ ou } (A \in 3 \text{ et } B \in 2) \\ 1_H = (A \in 0 \text{ et } B \in 2) \text{ ou } (A \in 2 \text{ et } B \in 0) \text{ ou } (A \in 1 \text{ et } B \in 3) \text{ ou } (A \in 3 \text{ et } B \in 1) \\ 2 = (A \in 0 \text{ et } B \in 3) \text{ ou } (A \in 3 \text{ et } B \in 0) \text{ ou } (A \in 1 \text{ et } B \in 2) \text{ ou } (A \in 2 \text{ et } B \in 1) \end{cases}$$

que l'on peut, par commodité, ramener à

$$\begin{cases} 0 = \text{idem} \\ 1_V \text{ ou } 2 = ((A \in 0 \text{ ou } 2) \text{ et } (B \in 1 \text{ ou } 3)) \text{ ou } ((A \in 1 \text{ ou } 3) \text{ et } (B \in 0 \text{ ou } 2)) \\ 1_H \text{ ou } 2 = ((A \in 0 \text{ ou } 1) \text{ et } (B \in 2 \text{ ou } 3)) \text{ ou } ((A \in 2 \text{ ou } 3) \text{ et } (A \in 0 \text{ ou } 1)) \end{cases}$$

Le schéma logique suivant implémente (partiellement) les opérations Ω_1 et Ω_2.

* *Note : Les cas aux limites (A ou B $\in D_H$ ou D_V) s'implémentent au choix par $\leq$ ou $\geq$*



L'évaluation parallèle de Ω_1 et Ω_2 nécessite un temps égal à

$$t_{\Omega_1} = t_c + 3t_e \quad \text{avec} \quad \begin{cases} t_c : \text{comparateur} \\ t_e : \text{porte logique} \end{cases}$$

(ce temps est déterminé par l'évaluation du cas 0).

On remarque donc que $t_{\Omega_1} - t_\sigma = t_e$.

Le circuit logique complet contiendrait 4 comparateurs et 19 portes logiques (remplaçables par une ROM 8→3 ou un PLA)

b) Ω_3

Cette opération sera réalisée à l'aide de deux circuits σ_3 en parallèle, l'un alimentée par α, l'autre par β.

$$t_{\Omega_3} = t_{\sigma_3}$$

3. Performances moyennes

Si p_i est la probabilité de ne pas avoir d'intersection, c'est à dire que A et B appartiennent au même quadrant, pour un niveau i, on obtient :

$$\bar{T}_{//} = t_{\Omega_1} * p_i + (t_{\Omega_1} + t_{\Omega_3}) * (1 - p_i)$$

$$\bar{T}_{//} = t_{\Omega_1} + (1 - p_i) \times t_{\Omega_3}$$

Comme $\begin{cases} t_{\Omega_1} \neq t_{\Omega_1} \\ t_{\sigma_3} = t_{\sigma_3} \end{cases}$

il vient $\bar{T}_{//} = t_{\sigma_1} + (1-p_i)\ t_{\sigma_3}$

La comparaison avec la solution série montre le net avantage de cette solution parallèle ($\bar{T}_S = 2t_{\sigma_1} + t_{\sigma_3}$).

Le gain moyen compris entre p_i et 50 % vaut

$$\bar{G} = \frac{\bar{T}_S - \bar{T}_{//}}{\bar{T}_S} = \frac{t_{\sigma_1} + p_i\ t_{\sigma_3}}{2t_{\sigma_1} + t_{\sigma_3}}$$

En ce qui concerne l'implémentation, la solution série met en jeu $C_S = 3 \times \sigma_1 + 3 \times \sigma_3$; la solution parallèle ne requiert que $C_{//} = \Omega_1 + 2\sigma_3$; si l'on considère que $\Omega_1 \simeq 2\sigma_1$, on obtient

$$\begin{cases} c_{//} = 2_{\sigma_1} + 2_{\sigma_3} \\ c_s = 3_{\sigma_1} + 3_{\sigma_3} \end{cases}$$

soit un gain de 1/3 pour la solution parallèle par rapport à la solution série.

4. Solution intégralement parallèle

Elle reviendrait à calculer les deux intersections (Ω_3) en parallèle avec l'évaluation de Ω_1, Ω_2 ; on ne retiendrait que celles nécessaires ; il est alors nécessaire que Ω_3 réalise l'intersection de la droite AB avec D_H et D_V.

Dans ce cas on aurait

$$\bar{T}_{//} = p_i\ t_{\Omega_1} + (1-p_i).t_{\Omega_3} \neq p_i\ t_{\sigma_1} + (1-p_i)\ t_{\sigma_3}$$

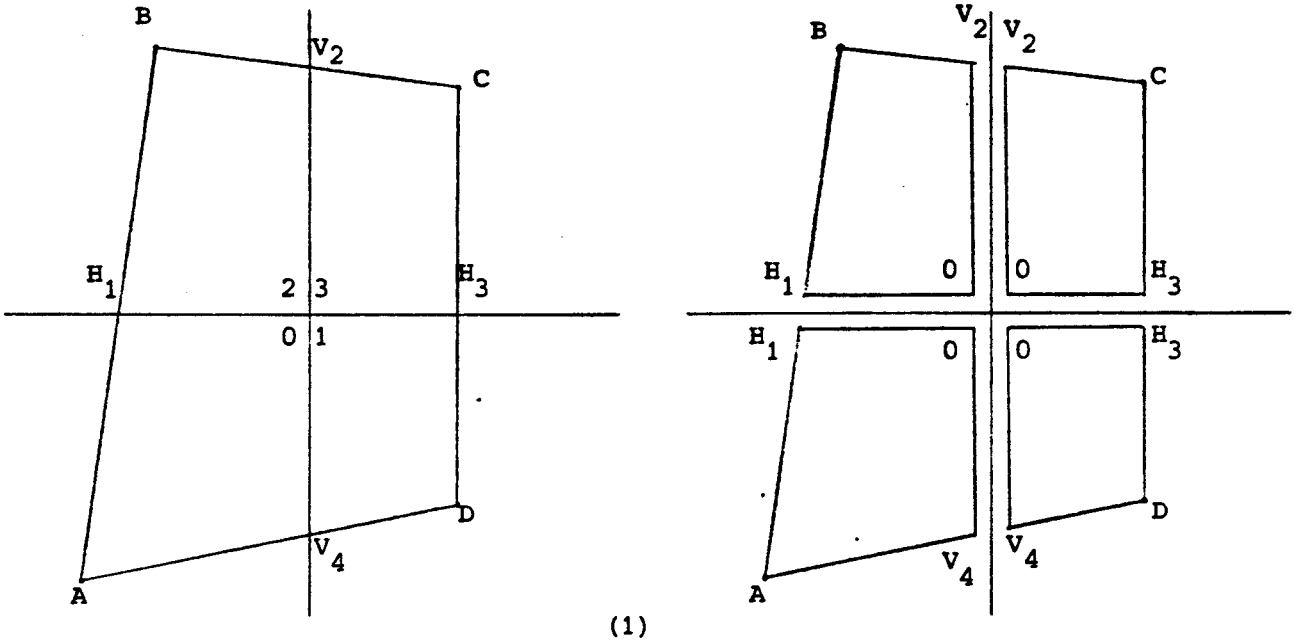
$$\text{et } G = \frac{2t_{\sigma_1} + t_{\sigma_3} - (p_i)\ t_{\sigma_1} - (1-p_i)\ t_{\sigma_3}}{2t_{\sigma_1} + t_{\sigma_3}}$$

$$\bar{G} = \frac{(2-p_i)\ t_{\sigma_1} + p_i\ t_{\sigma_3}}{2t_{\sigma_1} + t_{\sigma_3}} \quad \left(\text{compris entre } p_i \text{ et } 1 - \frac{p_i}{2}\right)$$

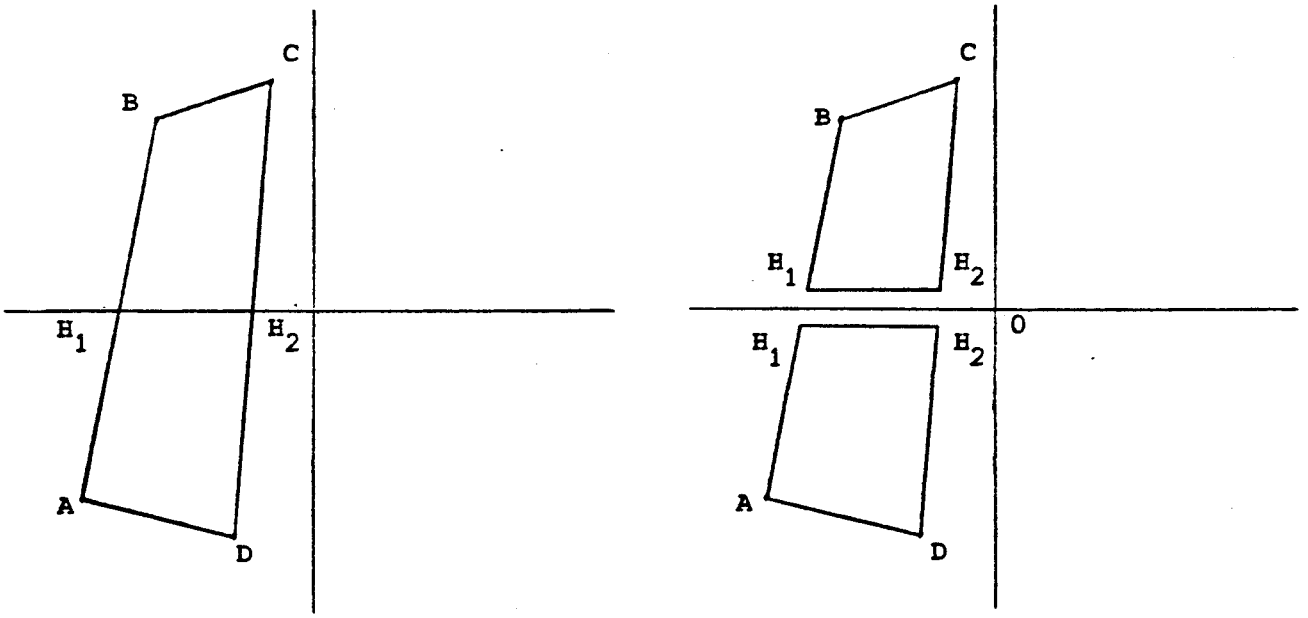
II.2.4.3 - Extension au découpage de polygones convexes

De même que Sutherland et Hodgman ont proposé une solution pour un découpage réentrant de polygone, nous étudions ici le même problème dans le cadre de notre découpeur bi-dimensionnel.

Donnons d'abord deux exemples des problèmes qui se produisent dans le traitement des polygones convexes :



(1)



(2)

La stricte application de l'algorithme de segments donnerait les résultats faux suivants :

cas (1)

fenêtre	AB	BC	CD	DA	résultat brut	résultat vrai
0	A H_1	-	-	V_4 A	A H_1, V_4 A	A H_1 0 V_4 A
1	-	-	H_3 D	D V_4	H_3 D V_4	H_3 D V_4 0 H_3
2	H_1 B	B V_2	-	-	H_1 B V_2	H_1 B V_2 0 H_1
3	-	V_2 C	C H_3	-	V_2 C H_3	V_2 C H_3 0 V_2

cas (2)

fenêtre	AB	BC	CD	DA	résultat brut	résultat vrai
0	A H_1	-	H_2 D	DA	A H_1, H_2 D A	A H_1 H_2 D A
1	-	-	-	-	-	-
2	H_1 B	BC	C H_2	-	H_1 B C H_2	H_1 B C H_2 H_1
3	-	-	-	-	-	-

Il est donc évident que les résultats bruts doivent être complétés pour produire les polygones exacts. Les règles à appliquer peuvent être formulées de deux manières équivalentes :

I - 1) Si le premier sommet d'une chaîne est un sommet original du polygone, la chaîne est fermée correctement ; dans ce cas il peut y avoir dans la chaîne une séquence HV (ou VH), qui doit être transformée en HOV (ou VOH).

2) Si le premier sommet est une intersection H (ou V), alors si le dernier est une intersection V (respectivement H), il faut fermer la chaîne par OH (resp. OV) ; sinon il faut fermer la chaîne par H (respectivement V).

D'où les règles de réécritures :

Si le polygone P initial est constitué de sommets A_i et que H_i et V_i sont les intersections trouvées, on obtient

1 $\begin{cases} A_i\ Q \rightarrow A_i\ Q \\ A_i\ Q_1\ H_j\ V_k\ Q_2 \rightarrow A_i\ Q_1\ H_j\ O\ V_k\ Q_2 \\ A_i\ Q_1\ V_j\ H_k\ Q_2 \rightarrow A_i\ Q_1\ V_j\ O\ H_k\ Q_2 \end{cases}$

$$2\begin{cases} H_i \, Q \, V_j \rightarrow H_i \, Q \, V_j \, O \, H_i \\ H_i \, Q \, H_j \rightarrow H_i \, Q \, H_j \, H_i \\ V_i \, Q \, H_j \rightarrow V_i \, Q \, H_j \, O \, V_i \\ V_i \, Q \, V_j \rightarrow V_i \, Q \, V_j \, V_i \end{cases}$$

II - 1) Si O est intérieur au polygone, alors O appartient à tous les polygones résultants, et se trouve entre le H et le V de chacun d'eux.

2) Sinon, O n'appartient à aucune chaîne.

Si nécessaire, il suffit de fermer la chaîne par répétition du 1er sommet en fin de chaîne.

Une caractérisation simple est donnée par

1) <=> il y a 4 polygones produits

2) <=> au moins une fenêtre est vide.

Il n'y a en réalité que trois cas

α : pas d'intersection, une seule fenêtre occupée

β : 2 intersections H (ou V), deux fenêtres occupées

γ : 4 intersections (2H et 2V), quatre fenêtres occupées.

L'application de ces règles simples permet d'obtenir le polygone exact en sortie.

II.2.4.4 - Applications aux problèmes posés

1. Algorithme de Warnock

Il convient à ce point de l'étude de vérifier que l'on satisfait bien à l'objectif fixé ; en particulier, dans le cas de l'algorithme de Warnock, on cherche à savoir si le polygone produit, qui, rappelons le, est intérieur à la fenêtre, est :

- inexistant (fenêtre vide)
- englobant : cela signifie simplement que la chaîne obtenue se réduit aux quatre sommets de la fenêtre
- intersectant : dans la chaîne obtenue se trouve un sommet différent des quatre sommets de la fenêtre.

Les informations en z, calculées simultanément, permettent alors de répondre aux questions que pose l'algorithme de Warnock.

2. Reconnaissance de formes et remplissage

Si les objectifs sont différents, ces deux applications nécessitent cependant des traitements identiques, résultant en la production d'un arbre quaternaire de présence.

II.2.5 - PERFORMANCES - EXTENSIONS

II.2.5.1 - Essai d'évaluation théorique

a hypothèses, calculs préliminaires

Nous supposons ici que nous disposons de découpeurs 2×2, de temps de traversée T_c par sommet ; nous examinons le traitement d'un polygone de k sommets, de périmètre p, à afficher pour une image $2^n \times 2^n$.

Le périmètre p est évalué comme étant le nombre de pixels finaux permettant d'inscrire le contour dans la mémoire ; dans le cas d'horizontales et de verticales, il est égal au périmètre mathématique.

Dans la suite, nous appellerons "sortie active" une sortie à intersection, nécessitant le passage au niveau suivant de découpage ; α_i sera le nombre de sortie active ($\star$ $0 \leq \alpha \leq 4$) et on posera $\beta_i = \alpha/4$, pour le niveau i.
Si l'on admet que les pixels finaux sont tous produits par le niveau 0 (ultime) de la récursion, ce qui est faux en général on obtient

$$\prod_{i=o}^{n-1} \beta_i = \frac{p}{2^{2n}}$$

Si nous supposons de plus que le processus récursif est statistiquement identique à lui-même, c'est à dire que $\beta_i = \beta$ = constante, on obtient

$$\beta^n = \frac{p}{2^{2n}}$$

soit $$\boxed{\alpha = p^{\frac{1}{n}}}$$

Ceci nous parait être une bonne approximation de la réalité ; une étude plus précise en fonction des polygones traités (taille, emplacement) s'avèrerait ici nécessaire.

b Par suite, le nombre de découpeurs utilisés à un niveau j est

$$\boxed{\alpha^j = p^{j/n}} \qquad \text{(en moyenne)}$$

Le nombre total de découpeurs à utiliser est donc égal à

$$\boxed{N = \sum_{j=o}^{n-1} \alpha^j} = \frac{\alpha^n - 1}{\alpha - 1} \qquad (\text{si } \alpha \neq 1)$$

Le tableau suivant résume les valeurs théoriques de α^j et de N obtenues pour différentes valeurs de p, dans l'hypothèse où n = 10 (image 1024×1024).

p	α	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9	N	
1	1	1	1	1	1	1	1	1	1	10	(1)
10	1.26	1.58	2	2.51	3.16	3.98	5.01	6.31	7.94	38	
100	1.58	2.51	3.98	6.31	10	15.85	25.1	39.81	63.1	171	
1024	2	4	8	16	32	64	128	256	512	1023	(2)
4096	2.30	5.27	12.12	28	64	147	338	776	1783	3159	(3)
10000	2.51	6.3	15.9	39.8	100	251	631	1585	3981	6615	
100000	3.16	10	31.6	100	316	1000	3162	10000	31623	46248	
1M	4	16	64	256	1024	4K	16K	64K	256K	341K	(4)

(Pour le calcul de N, les valeurs de α^j ont été arrondies aux entiers supérieurs).

(1) dans ce cas, très académique, chacun des dix découpeurs a une seule sortie active

(4) ce second cas, lui aussi très académique, se caractérise par un arbre de découpeurs complet

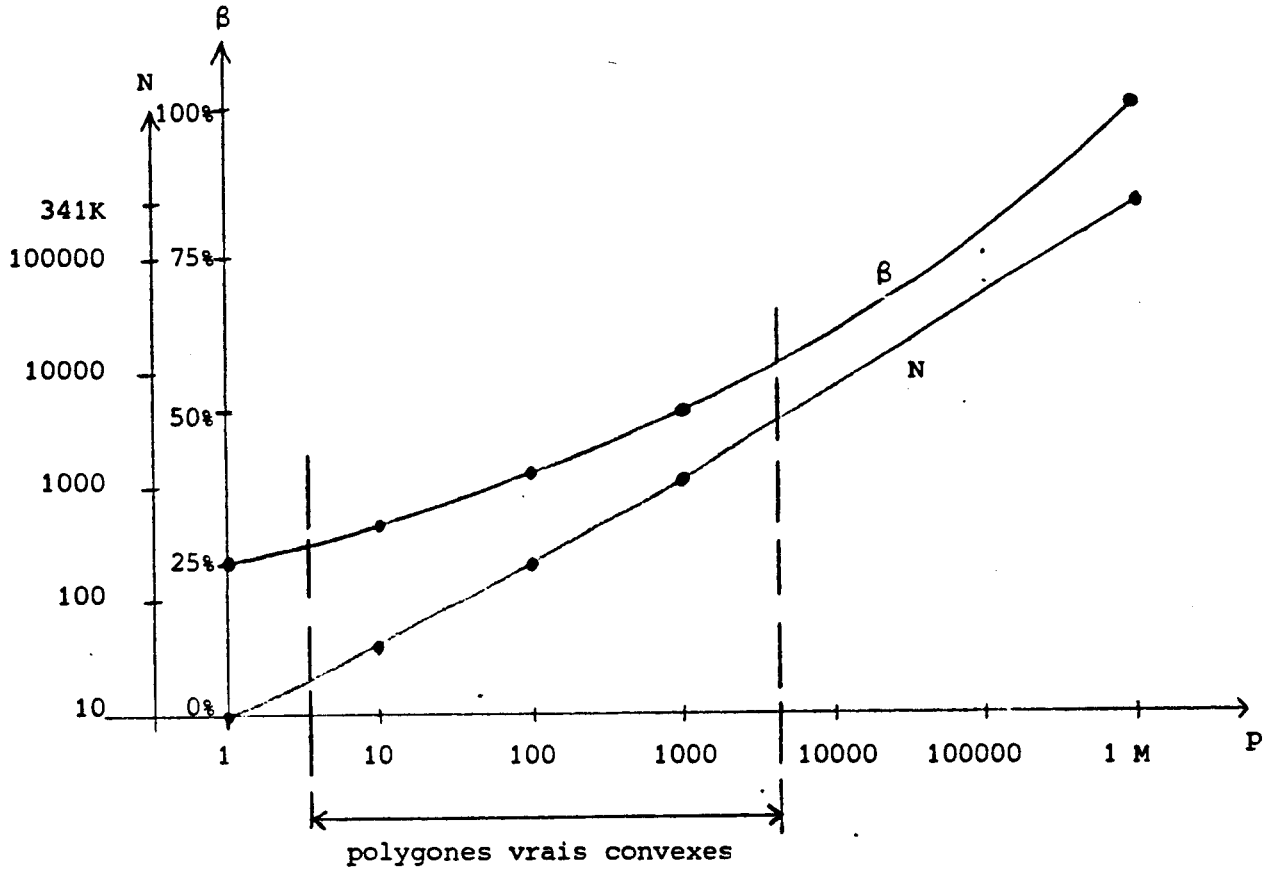
(2) ce cas correspond à un usage à 50 % de tous les découpeurs ; l'arbre quaternaire se transforme en arbre binaire, avec les propriétés associés (notamment : un niveau i comporte autant de découpeurs que l'ensemble des niveaux supérieurs)

(3) le périmètre p est égal au périmètre de l'écran : c'est le plus grand polygone convexe

Les courbes suivantes donnent les variations de

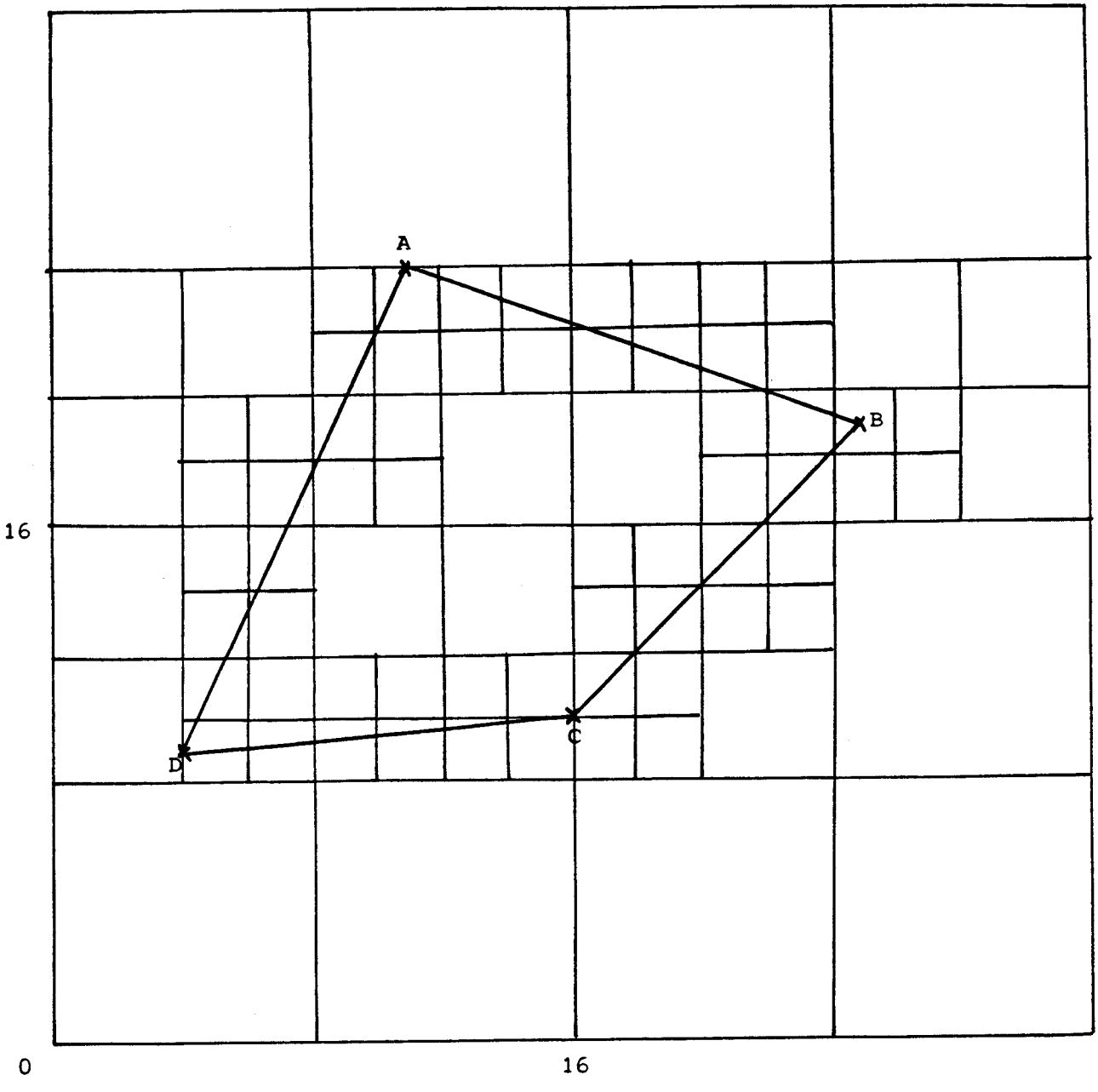
. β en fonction de p (coordonnées semi-logarithmiques)

. N en fonction de p (coordonnées logarithmiques).



II.2.5.2 - Exemple

L'exemple suivant, sur une image $2^5 \times 2^5$ illustre et vérifie la théorie ci-dessus.



$$\begin{cases} A = (11, 24) \\ B = (25, 19) \\ C = (16, 10) \\ D = (4, 9) \end{cases} \qquad \begin{cases} AB = 14 \\ BC = 9 \\ CD = 12 \\ DA = 15 \end{cases}$$

périmètre = 50

d'où $\begin{cases} \alpha = 2.19 \text{ arrondi à } 3 \\ \beta = 54{,}7\ \% \end{cases}$

et N = 42

De plus $\alpha^2 = 5$

$\alpha^3 = 11$

$\alpha^4 = 23$

Le tableau suivant résume l'analyse manuelle de l'image.

				total			
niveau	n	pleins	vides	n	pleins	vides	% total général
1	4	0	0	4 100 %	0	0	100 %
2	2 1 2 2	0 0 0 0	2 3 2 2	7 44 %	0	9	7/16 = 44 %
3	2 2 3 1 3 3 1	0 2 0 0 1 1 0	2 0 1 3 0 0 3	15 54 %	4	9	15/64 = 23 %
4	3 2 3 2 1 1 1 1 2 3 2 3 1 2 1	1 2 0 2 0 3 0 0 0 1 2 0 1 2 0	0 0 1 0 3 0 3 3 0 1 0 1 3 0 3	28 47 %	14	18	28/256 = 11 %
5	etc			54 48 %			54/1024 = 5,3 %
correspondance théorique			α^i et β				β^i

Le tableau suivant compare les valeurs théoriques et pratiques de p, α^i, β^i et N.

	théorie	pratique
p	50	54
α	3	4
α^2	5	7
α^3	11	15
α^4	23	28
β	54,7 %	100 %
β^2	30 %	44 %
β^3	16 %	23 %
β^4	9 %	11 %
N	43	55

et aussi 44 %, 54 %, 47 % et 48 %

Cette comparaison montre à l'évidence la validité de la théorie surtout dans les bas niveaux de l'arbre de récursion.
Notons enfin que cet exemple simple correspond à un polygone de périmètre 2500 dans une image $2^{10}\times 2^{10}$; nous aurions alors eu N = 964.

II.2.5.3 - *Performances*

Elles dépendent fortement de la possibilité de réaliser un pipe-line entre les niveaux ; si c'est le cas, le temps total de traitement est égal à :

$$t_o = \underbrace{(k-1)\times T_c}_{\text{mise en charge}} + \underbrace{nT_c}_{\text{traversée du pipe-line pour un sommet}} = (k+n-1)\times T_c$$

en supposant que l'on doive traverser tous les niveaux de récursion.

Si ce n'est pas le cas, le temps total dépend du nombre de pipe-lines devant être traités séquentiellement ; si l'on ne peut pas du tout opérer en pipe-line, on obtient

$$\boxed{t_1 = n \times k \times T_c}$$

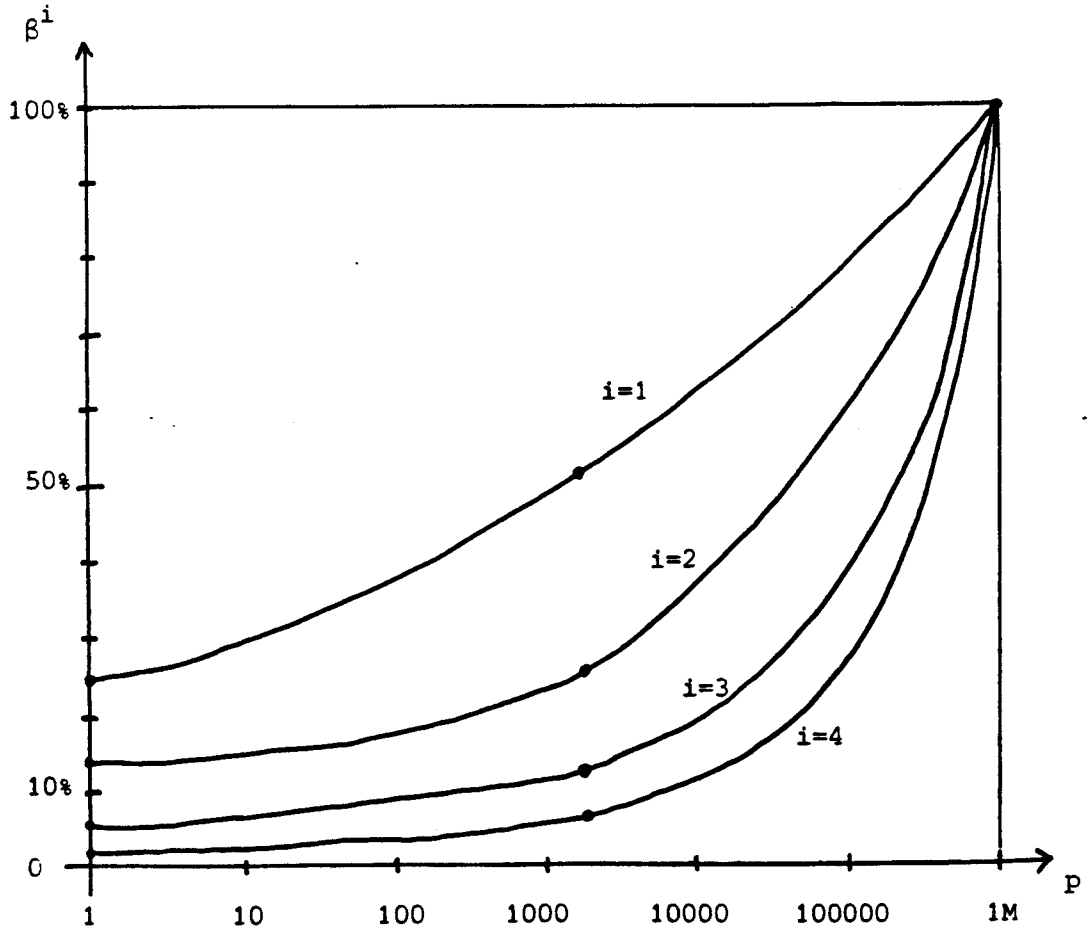
Ces deux cas supposent que l'on soit capable de traiter en parallèle l'ensemble d'un niveau, en particulier le niveau le plus bas ; il est donc nécessaire de disposer de α^{n-1} découpeurs dans cette hypothèse ; on peut alors affirmer que $t_o < t < t_1$.

i	sorties	nombre de découpeurs 2×2	T/T_c par sommet	nombre de récursions	intérêt	faisabilité technologique
1	4	1	1	10	*	***
2	16	5	2	5	*	**
3	64	21	3	4 (1)	*	*
4	256	85	4	3 (1)	**	?
5	1K	341	5	2	***	-
6	4K	1365	6	2 (1)		-
7	16K	5461	7	2 (1)		-
8	64K	21K	8	2 (1)		-
9	256K	85K	9	2 (1)		-
10	1M	341K	10	1	***	!!

(1) arrondi à l'entier supérieur : le nombre de niveaux réels traitable est égal au nombre de récursions ×i.

Nous ferons simplement les remarques suivantes :

1) le cas i = 10 est une hypothèse d'école, mais très séduisante
2) seules β^i sorties sont utilisées ; la courbe suivante indique le rendement obtenu en fonction du périmètre des polygones traités et du nombre de niveaux intégrés.



Dès que P > 1024, on a $\sum_{i=o}^{n-2} \alpha^j = \frac{\alpha^{n-1}-1}{\alpha-1} < \alpha^{n-1}$, c'est à dire que si l'on dispose de α^{n-1} découpeurs, on peut assurer globalement les niveaux i < n-1. Le temps obtenu est alors

$$t_2 = (k+n-2) \times T_c + kT_c = (2k+n-2)\ T_c$$

Si l'on ne dispose pas des α^{n-1} découpeurs, le temps d'exécution se dégrade car on ne peut plus assurer le parallèlisme dans un niveau donné.

Le tableau suivant résume les valeurs de t_o, t_1, t_2 dans l'hypothèse où n = 10, pour quelques valeurs de k

k	t_o/T_c	t_1/T_c	t_2/T_c
3	12	30	14
4	13	40	16
6	15	60	20
10	19	100	38

Ainsi, pour T_c = 100 ns, dans l'hypothèse optimale t_o, on aurait

$$t_o \simeq 1{,}5\ \mu s \quad \text{si } \bar{k} = 6\ ;$$

la machine est donc capable de traiter plus de 2500 polygones par trame de balayage (40 ms).

Notons cependant ici que toutes ces évaluations dépendent beaucoup de l'algorithme et de la "micro-architecture" associée. Notre propos n'était pas ici de discuter de l'organisation et du contrôle de ces composantes de base, problèmes certes importants, mais connus par ailleurs ; il est certain qu'une machine récursive pose de gros problèmes de mémorisation et de contrôle, mais ceux-ci ne relèvent pas spécifiquement de l'informatique graphique. Aussi les avons-nous volontairement laissés de côté dans une étude centrée sur les algorithmes graphiques et les composants de base.

II.2.5.4 - *Extensions*

Nous proposons deux extensions

α) L'une, ne posant que des problèmes techniques, consiste à intégrer non pas un seul découpeur, mais un arbre complet couvrant i niveaux, c'est à dire un découpeur $2^i \times 2^i$. Le tableau suivant résume quelques caractéristiques d'un tel découpeur pour une image $2^{10} \times 2^{10}$:

3) Si on peut penser que l'on saurait intégrer jusqu'au niveau i = 3 (peut-être 4), la construction câblée d'un arbre de niveau supérieur peut être intéressante.

Restent les questions ouvertes suivantes ?

. faut-il intégrer le plus possible, au prix d'un faible taux d'utilisation ?

. faut-il créer de grands arbres de découpeurs ou plutôt disposer de nombreux découpeurs de base, dont la gestion paraît délicate ?

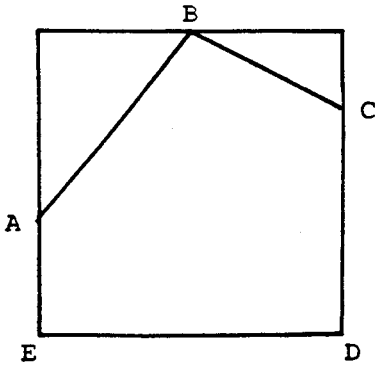
β) L'autre, plus théorique, consiste à réitérer notre modèle 2×2 pour obtenir un modèle 4×4, voire plus ; l'étude reste à faire, et en première approche, un modèle 4×4 semble possible à réaliser.

II.2.6 - APPLICATION AU TRAITEMENT DU SOUS-ECHANTILLONAGE

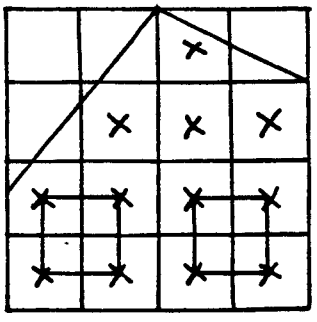
Une méthode par sur-résolution est tout à fait envisageable. Elle consiste à poursuivre le processus de découpage proprement dit jusqu'à un niveau plus petit que le pixel ; on obtient ainsi pour un pixel donné, un arbre de présence dans ses "sous-pixels".

Exemple

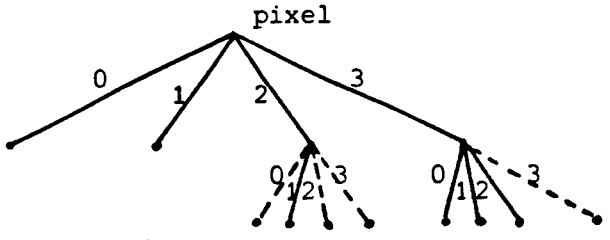
Sur-résolution 4.



polygone théorique



résultat obtenu



poids 4

poids 1

Cette méthode suppose d'effectuer les calculs à la précision de la sur-résolution ; le découpage ultime fournit ainsi un arbre, interprétable de deux manières.

1) Simple, la contribution du polygone au pixel étant la somme des sous-pixels (ici $2\times4 + (1+3) \times 1 = 12$, à normaliser par rapport à $4^2 = 16$, soit $s = 12/16 = 75$ %).

2) Plus complexe, par filtrage bidimensionnel, i.e. en attribuant des poids différents aux sous-pixels suivant leur emplacement dans le pixel.

Exemple : filtrage triangulaire (x, y)

{ les sous-pixels du centre ont le poids 2

{ les sous-pixels du pourtour ont le poids 1

Le pixel a donc un poids total 20.
Un pavé 2×2 a un poids (2+3) = 5
et l'on obtient ici une contribution s'

$$s' = \frac{2\times5+2\times2+2\times1}{20} = \frac{16}{20} \text{ soit } 80\ \%$$

Notons pour conclure que cette dernière méthode est la plus proche du traitement "naturel" du sous échantillonnage (utilisé en traitement d'images), mais ne se justifie pas en synthèse (car on n'a pas à y corriger les défauts du capteur).

CONCLUSION

Il est bien évident que l'étude a été menée sur l'exemple du découpage spatial, mais des études analogues devraient l'être sur tous les algorithmes classiques de synthèse d'images, tels que :

. l'algorithme du z-buffer

. les interpolateurs (Bresenham ...) voir ([DUR 82]).

. Newell, Newell et Sancha, par exemple.

L'exemple traité a simplement la prétention de montrer trois points :

. l'importance de l'analyse précise et complète des algorithmes, sans les contraintes des machines usuelles

. l'influence de cette analyse sur l'architecture d'une machine adaptée à chaque algorithme, et aussi sur les composants à réaliser pour cette architecture

. l'intérêt de poser les problèmes à un niveau relativement élémentaire, en raison de la faisabilité sous forme simple (PLA, PROM, automates) des opérateurs correspondants judicieusement choisi.

Les problèmes bien connus de contrôle et de mémorisation doivent bien entendu être traités également pour cet ensemble d'opérateurs spécifiques.

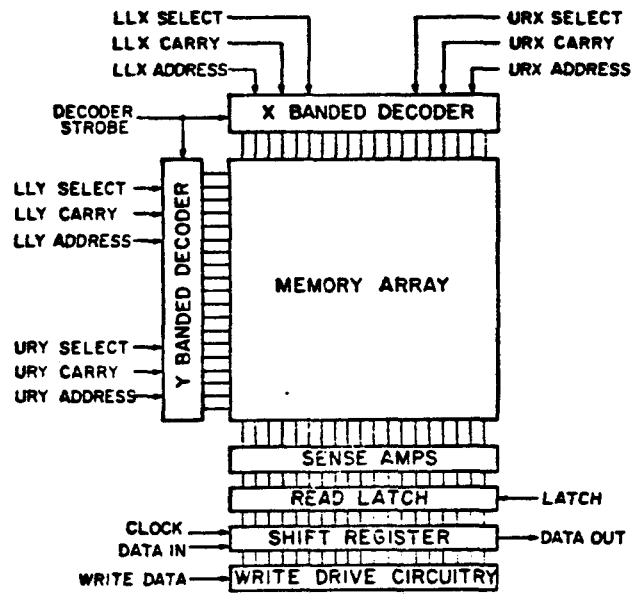
Notons un dernier point : il se trouve que l'algorithme choisi déboucherait sur la réalisation de composants tous identiques, ou presque ; mais cette observation est un leurre : en réalité à chaque algorithme correspond son ou ses composants, et l'on ne saura produire ou traiter une image que si l'on dispose d'un nombre minimal de ces différents composants. C'est là l'inconvénient majeur des machines de type algorithme : leur réalisation passe par des composants différents et complexes ; aussi restons-nous très réservés quant à l'avenir de cette classe d'architectures.

ANNEXE : Mémoire de Trame à remplissage par pavés

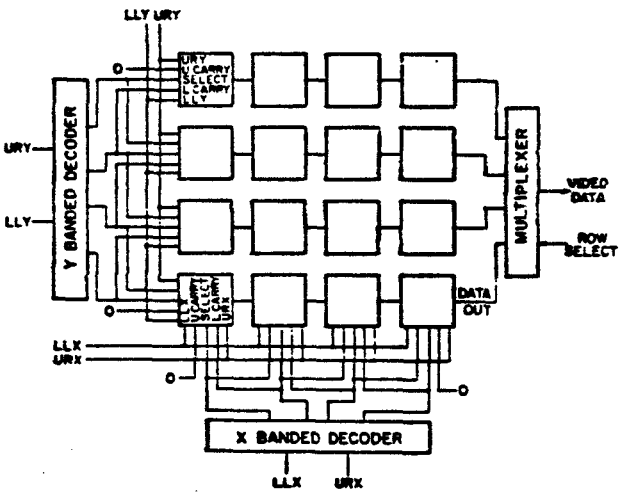
L'étude de l'algorithme de Warnock et du remplissage par découpage a montré qu'un des éléments de ces problèmes est finalement de remplir une zone de la mémoire d'image à l'aide d'une valeur constante.

1. Cas d'une zone rectangulaire : mémoire de Whelan

Whelan ([WHE 82] a proposé une architecture permettant le remplissage de zones rectangulaires à bords parallèles aux axes. Le composant qu'il propose assure ce remplissage automatiquement par écriture en cascade entre les limites x_{min}, x_{max} d'une part, y_{min}, y_{max} d'autre part qu'il reçoit.



De tels composants peuvent être combinés pour réaliser une mémoire de trame complète.



Whelan annonce (1982) la conception d'un composant à 4096 pixels (1 bit/pixel) en n-MOS ; une mémoire complète pour une image 1024×1024×8, contiendrait donc 2048 de ces composants.

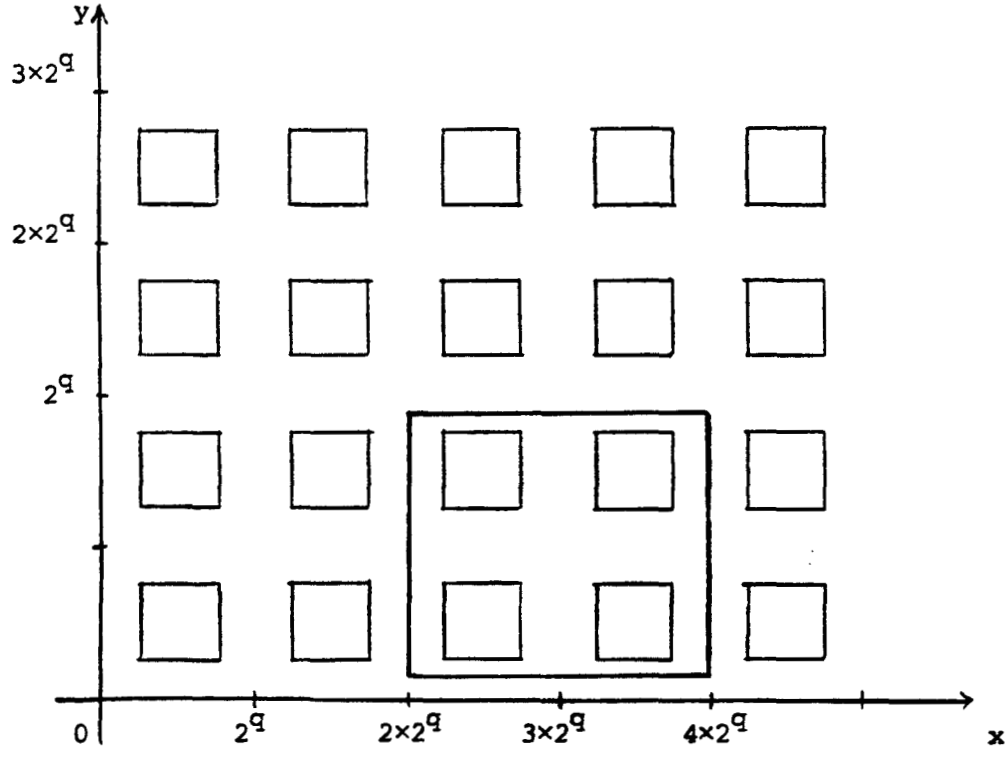
2. Cas particulier : zones carrées bien organisées

Dans les processus de remplissage concernés, on a toujours affaire à une zone carrée telle que :

$$x_{min} = X.2^p$$
$$x_{max} = (X+1).2^p-1$$
$$y_{min} = Y.2^p$$
$$y_{max} = (Y+1).2^p-1$$

Si nous supposons que le composant élémentaire a une taille $2^q \times 2^q$, deux cas sont à envisager.

a) $p \geq q$: le remplissage concerne $4^{(p-q)}$ composants disposés en carrés, qu'il suffit de sélectionner en parallèle et de remplir totalement.



$$x_{min} = 2^{q+1}$$
$$x_{max} = 2\times2^{q+1}-1$$
$$y_{min} = 0$$
$$y_{max} = 2^{q+1}-1$$

soit $\begin{cases} X = 1 \\ Y = 0 \\ p = q+1 \end{cases}$

b) $p < q$: un seul composant est concerné ; il faut

- le sélectionner (extérieurement)
- réaliser le remplissage partiel

Pour ce faire il suffit de connaître x_{min}, y_{min}, et la taille de la zone à remplir, indiquée par p, ce qui ne pose pas de problème majeur.

Nous proposons donc une mémoire à remplissage par pavés carrés de taille 2^i, avec $0 \leq i \leq q$, simplification de la mémoire de Whelan. Une étude plus précise de faisabilité reste à faire, mais la technologie actuelle permettrait de réaliser des mémoires statiques adéquates de grande taille.

BIBLIOGRAPHIE

a) Composants et machines

[BAS 83], [CLA 80], [CLA 82], [EFCXX], [FER 81], [LER 78], [LER 80], [MAR 81], [MAR 82], [MER 79], [MER 84], [NECXX], [WHE 82].

b) Algorithmes

[BOU 80], [BRE 65], [CAT 79], [CEA 82], [CRO 77], [DUR 83], [ENC 75], [GRI 75], [LOC 80], [LUC 77], [PEL 84], [SPS 68], [SUH 74], [SUS 74], [WAR 69].

III - MACHINE-PIXEL

INTRODUCTION

Cette classe de machine est caractérisée par l'association d'un processeur logique à des zones de l'image obtenues par découpage géométrique. L'étude qui suit pousse à l'extrême ce découpage et l'on y suppose qu'à chaque pixel est associé un processeur. Nous avons indiqué et nous le montrerons dans la suite qu'une telle approche favorise un haut degré de parallélisme et se traduit matériellement par la réalisation de composants nombreux certes, mais tous identiques.

Après avoir rappelé les principes généraux des machines cellulaires, l'étude de quelques algorithmes remarquables nous conduira à bâtir une architecture cellulaire théorique, sur laquelle nous indiquerons comment l'on peut implémenter de nombreux algorithmes communs de traitement et de synthèse. Les aspects simulation et réalisation effective indiqueront un ordre de grandeur des performances que l'on peut attendre de ce type de solution.

III.1 - LES MACHINES CELLULAIRES

III.1.1 - DEFINITIONS ET PROPRIETES ([COD 68])

Il n'est pas ici dans notre propos de refaire l'historique des machines cellulaires. Néanmoins il n'est pas inutile d'effectuer quelques rappels.

Le concept de cellularité s'est exprimé dans deux principaux courants de recherches, les automates cellulaires et les tableaux logiques cellulaires ; la caractéristique commune est que ces machines sont constituées de cellules élémentaires toutes identiques et reliées entre elles.

a) Les tableaux logiques cellulaires.
Comme leur nom l'indique, ces tableaux sont constitués de cellules logiques, comportant chacune :

* des entrées internes (venant des cellules voisines) ou externes (venant de l'extérieur du réseau)
* des sorties internes ou externes
* une fonction logique, le plus souvent combinatoire, reliant les sorties aux entrées
* éventuellement une fonction mémoire.

Le tableau ci-dessous résume les caractéristiques de quelques réseaux logiques cellulaires :

réseau	nombre d'entrées	nombre de sorties	mémoire	fonctions réalisées
Maîtra	2	1	non	1
Minnick	2	2	non	16
Ackers	3	2	non	1
Kautz	3	3	oui	1
Duff	8	7	non	256

b) Les automates cellulaires
La notion d'automate cellulaire a été introduite vers 1950 par Von Neumann, qui a proposé un automate cellulaire universel a 29 configurations, i.e. pouvant évaluer toute fonction calculable (au sens de Türing). L'approche réalisée est plus formelle que pour les tableaux logiques.

* Un automate cellulaire repose sur un **espace cellulaire** ou ensemble de cellules réparties régulièrement dans un espace de dimension d.

Il existe une mesure discrète du temps appelée la suite des générations. Chaque cellule peut posséder à une génération donnée, un état choisi dans un alphabet A fini et non vide ; un état dit de repos, q_o est caractérisé par le fait qu'il est inchangé si les cellules voisines sont dans l'état q_o.

On appelle **configuration** l'ensemble des états de toutes les cellules à une génération donnée.

Le **voisinage** d'une cellule est un ensemble fini de cellules connectées directement.

La **fonction de transition** F est la loi permettant de déterminer l'état d'une cellule à partir

- de son état
- de l'état de ses voisines

à la génération précédente. Elle est la même pour toutes les cellules ; si elle est univoque, l'automate est dit déterministe.

* Citons quelques exemples :

→ automate de Von Neumann

. pavage carré du plan
. deux états : habité - inhabité
. voisinage : 4 cellules
. fonction de transition : la cellule devient habitée si une et une seule voisine l'était.

→ automate de Conway (Jeu de la Vie)

. pavage carré du plan
. deux états : habité - inhabité
. voisinage : 8 cellules
. fonction de transition : si la cellule est habitée, elle survit si deux ou trois voisines sont habitées ; si elle est inhabitée, il y a génération spontanée si trois voisines sont habitées.

→ automate de Nourai et Kashef

. pavage carré du plan
. quatre états : 0, 1, 2, X (0 = repos)
. voisinage : 4 cellules
. fonction de transition : c'est un ensemble de 43 règles de transition que l'on peut présenter sous forme de sixtuple ordonné associant le nouvel état à l'ancien état de la cellule et aux anciens états de ses 4 voisines.

c) Notion de machine cellulaire

La distinction entre réseau logique et automate cellulaire s'estompe très rapidement : une cellule contenant de la logique séquentielle n'est rien d'autre qu'un automate ; de même un automate à un seul état peut être vu comme une cellule de base d'un réseau logique.

Ainsi nous appelerons machine cellulaire un ensemble de modules élémentaires de traitement et de mémorisation, tous identiques, appelés cellules. Ces cellules sont reliées à un nombre fini de voisines, de manière uniforme. Le maillage ainsi obtenu est donc régulier. Chaque cellule dispose de :

- . entrées internes et externes
- . sorties internes et externes
- . mémoire (ou registres)
- . moyen de calcul élémentaire (Unité Arithmétique et Logique par exemple)
- . automate (par exemple µ-programme)

Chaque cellule peut se trouver dans un état parmi n états possibles. A chaque état correspond une séquence d'opérations élémentaires.

d) Fonctionnement

On peut distinguer deux types de fonctionnement :

* d'une part un mode SIMD, c'est à dire que toutes les cellules exécutent en même temps la même commande provenant d'une origine unique : le contrôle, la commande et la synchronisation sont assurés hors du réseau

* d'autre part un mode MIMD, où chaque cellule est autonome et exécute son propre programme ou micro-programme (contrôle réparti, local).

Entre ces deux extrêmes, on pourra trouver un fonctionnement macro-SIMD (ou micro MIMD), où toutes les cellules exécutent le même programme (ou microprogramme), mais pas obligatoirement la même instruction (ou micro-instruction).

Un autre type de fonctionnement peut être envisagé, analogue au pipe-line unidimensionnel, que nous appellerons "propagation" : l'activité d'une cellule dépend de l'activité de ses voisines ; dans ce cas le contrôle peut être soit centralisé, soit complétement propagé.

D'où 5 modes de fonctionnement :

- . micro SIMD
- . MIMD vrai
- . macro SIMD

. propagation à contrôle central

. propagation à contrôle local (propagé)

L'étude de quelques algorithmes au §III-2 nous permettra d'effectuer un choix parmi ces modes de fonctionnement.

e) Remarque

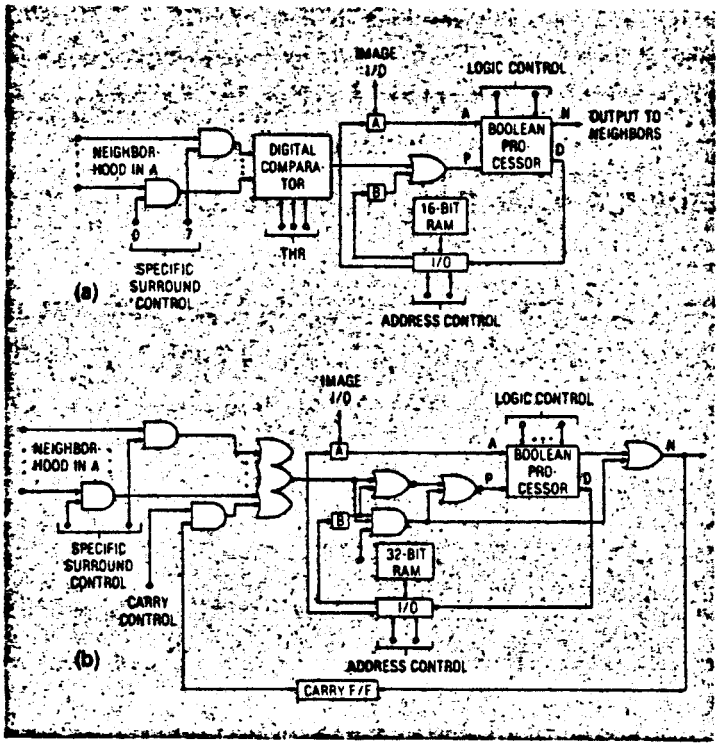
On notera que les architectures systoliques n'ont pas été évoquées ici, car, dans l'état actuel des travaux de recherches, ces machines sont spécialisées pour un algorithme précis ou pour une classe d'algorithmes dans lesquels les flots de données ou d'instructions se croisent de façon régulière et identique.

III.1.2 - QUELQUES MACHINES CELLULAIRES DE TRAITEMENT D'IMAGES

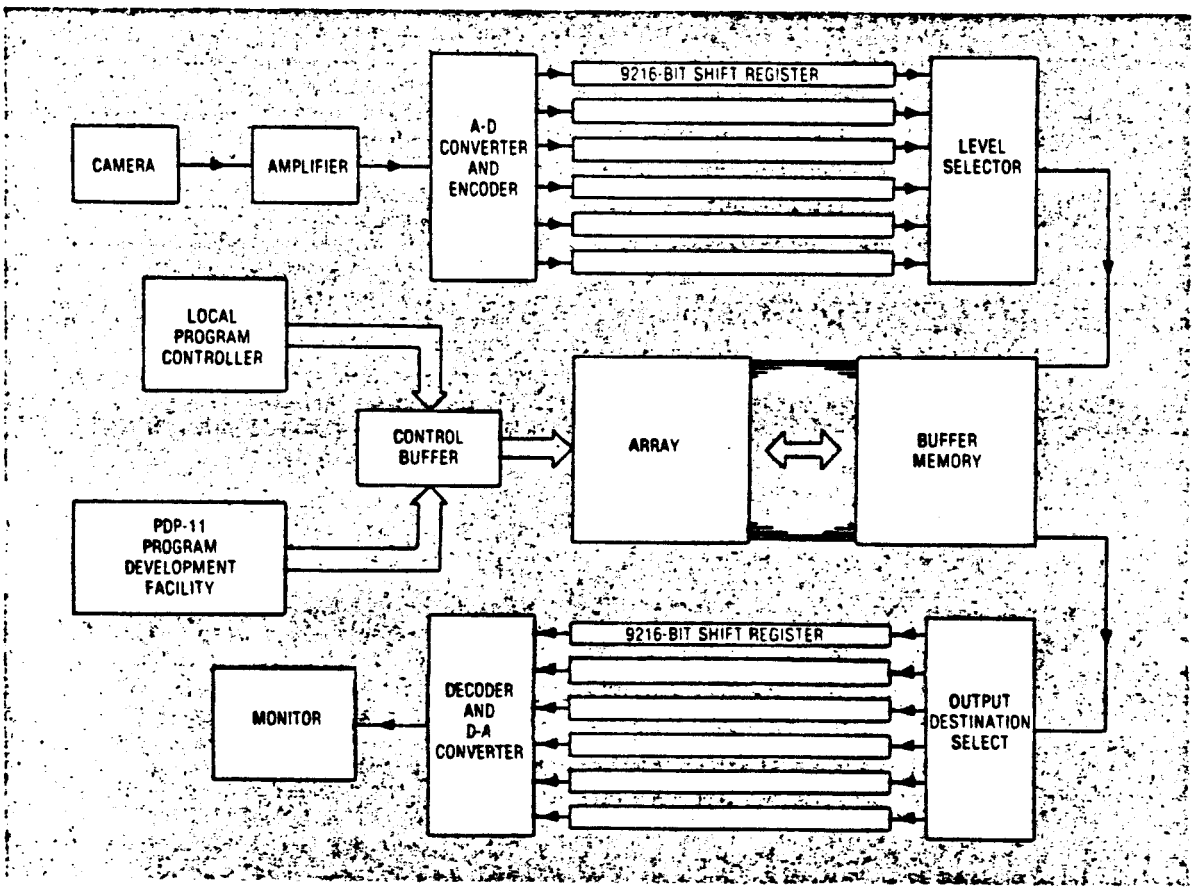
Depuis que S.M. Unger suggéra, en 1958 (!), l'utilisation de tableaux de processeurs comme approche naturelle pour le traitement d'images et la reconnaissance des formes, de nombreuses machines ont été étudiées et réalisées. On trouvera dans [ROS 83] et [KAI 83] des études exhaustives de ces machines. Nous nous limiterons ici à présenter succinctement trois de ces processeurs, CLIP, MPP et AAP.

a) CLIP (Cellular Logic Array for Image Processing) est le nom générique d'une série de machines cellulaires SIMD développées par DUFF, WATSON et al. CLIP4, le dernier en date, est une machine logique cellulaire formée de 96×96 cellules travaillant bit par bit, chaque cellule ayant 8 voisines possibles ([DUW 73], [PRE 83]).

L'architecture d'une cellule est la suivante :



La machine complète a l'architecture suivante :

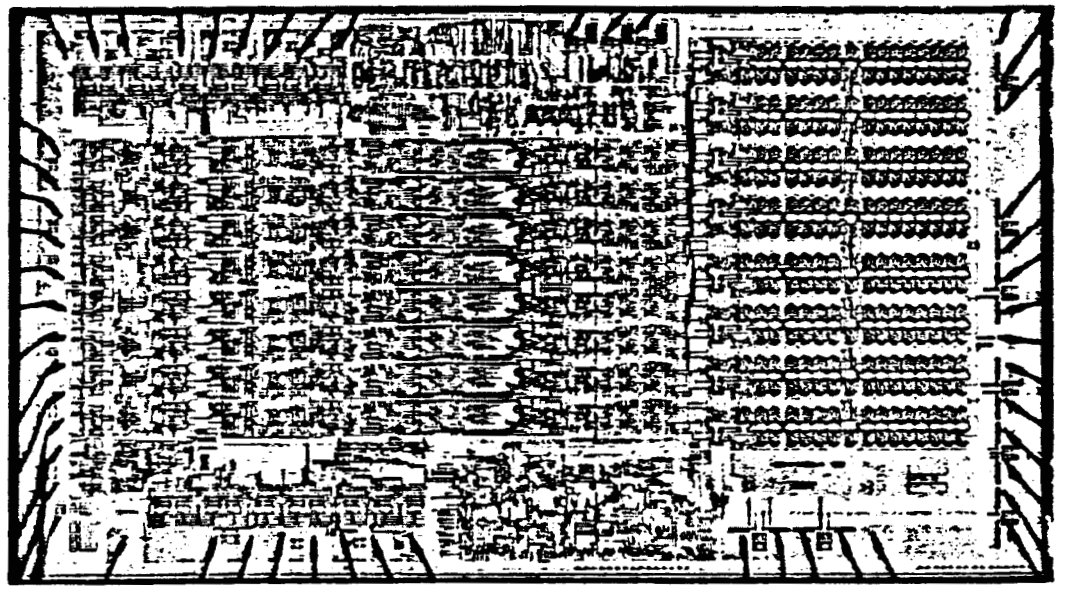


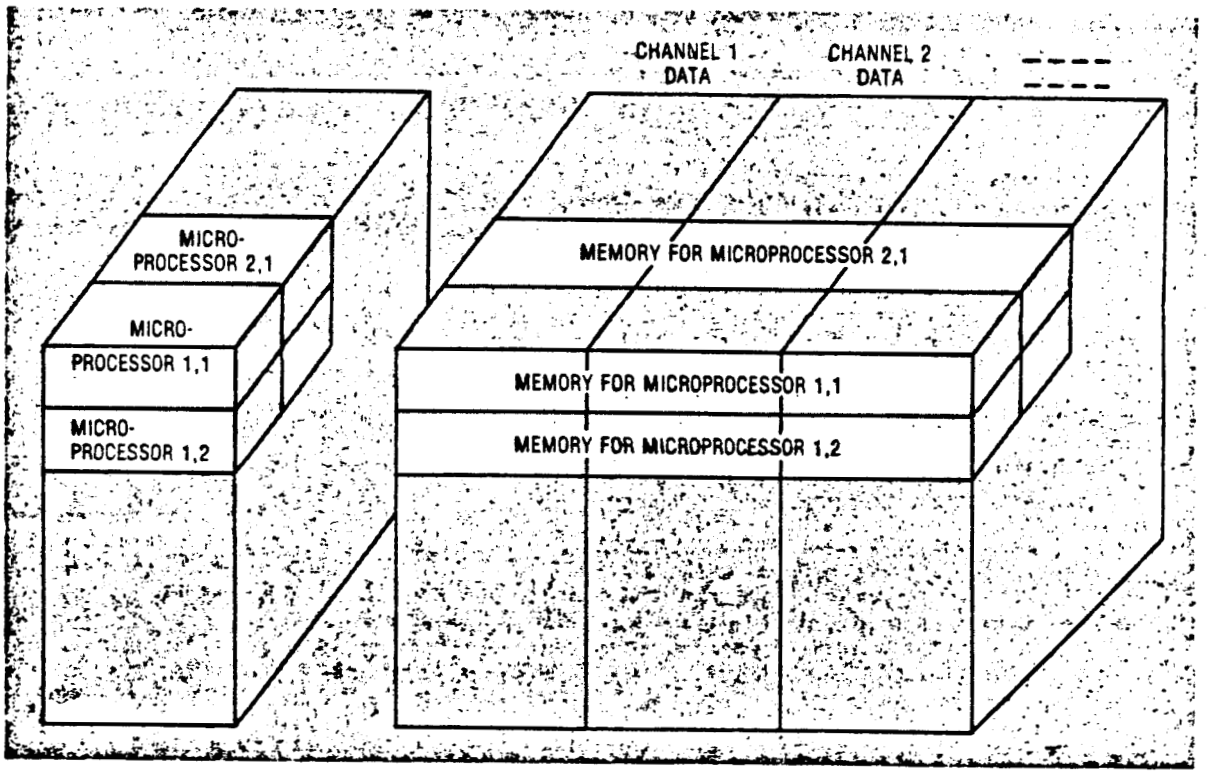
La puissance de calcul est d'environ 1000 Mips (compté en opérations élémentaires).

b) MPP (Massively Parallel Processor) est une machine SIMD comportant 16 K processeurs, construite pour la NASA. Cette machine peut être configurée en tableau de 128×128 processeurs. Chaque processeur peut accéder aléaoirement à une mémoire commune ; il est relié à quatre voisins. Chaque processeur est un microprocesseur puissant, dont le temps de cycle est de 100 ns. ([BAT80]). Les performances globales sont très élevées :

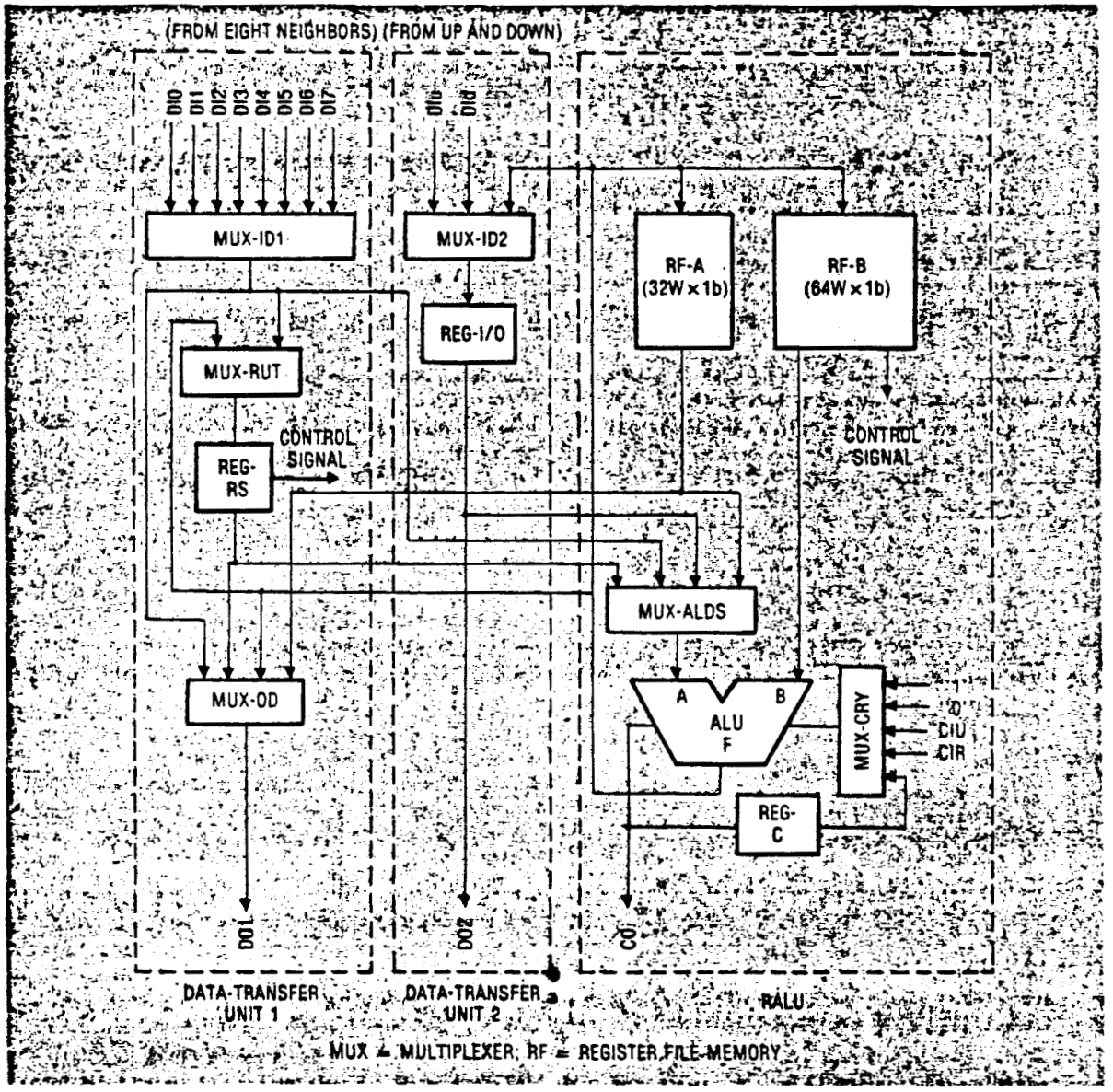
addition d'entiers (8 bits) : 6500 MOPS
multiplication de tableaux d'entiers : 1860 MOPS
convolution 7×7 : 14 MOPS

Cette machine est destinée au traitement d'images de satellites.





c) AAP (Adaptative Array Processor) est une machine SIMD formée de composants à haut degré d'intégration comprenant 8×8 processeurs travaillant bit par bit. Chaque cellule est reliée à 8 voisins. L'architecture d'une cellule est la suivante :



L'étude (succincte) de ces machines montre, ce que l'on retrouvera plus loin, que le traitement d'images utilise des algorithmes très parallèles et donc induisent une architecture SIMD, avec peu de communications.

III.1.3 - UNE MACHINE CELLULAIRE DE SYNTHESE D'IMAGES ([FUP 81])

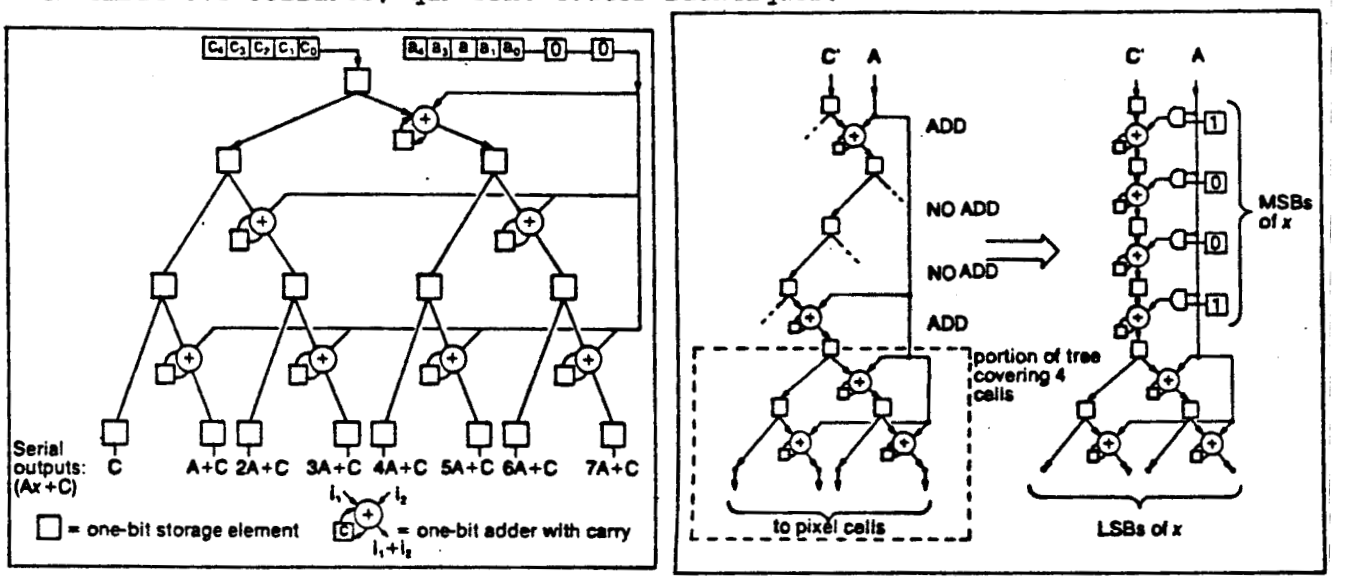
Avant d'aborder le problème sous l'angle algorithmique, il convient de mentionner l'une des rares machines cellulaires de synthèse d'images qui ait été proposée. Cette machine, proposée en 1981 par Fuchs et Poulton,

implémente le calcul de F(x, y) = Ax + By + C.

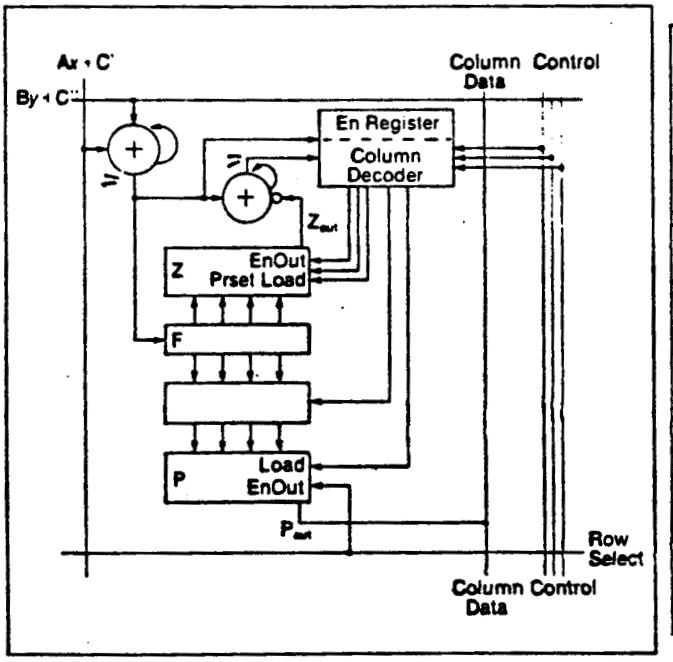
En effet, on peut montrer que l'évaluation de cette expression permet de déterminer :

- la visibilité (algorithme du z-buffer)
- l'ombrage (algorithme de Gouraud)
- le remplissage de polygones convexes (voir remplissage en parallèle).

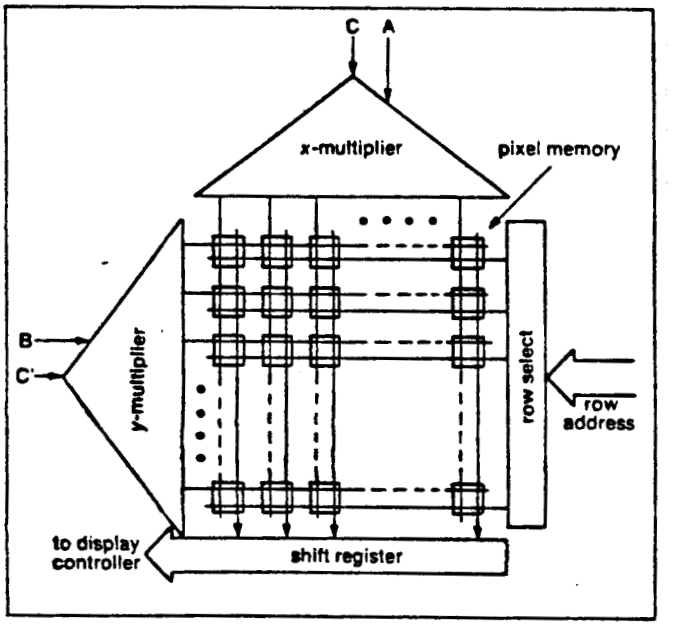
Le calcul de F(x, y) est réparti à l'aide d'arbres de multiplication dans l'ensemble des cellules, qui sont toutes identiques.



Arbre de multiplication : principe et répartition dans chaque cellule



La cellule élémentaire



La machine complète

Un préprocesseur élabore les valeurs de A, B, C en fonction de l'opération à effectuer. Les auteurs pensent que la machine (actuellement intégrée à raison de 4 cellules par composant) serait capable de traiter 15 à 30000 polygones par seconde.

Notons que, si l'on étudie de près l'implémentation du calcul de F, on constate que le calcul se fait par propagation, même si ce calcul est asynchrone dans l'architecture proposée. Nous constaterons que d'autres algorithmes de synthèse conduisent au même mode de fonctionnement.

III.2 - ETUDE DE QUELQUES ALGORITHMES DE SYNTHESE ET DE TRAITEMENT D'IMAGES

Nous étudions ici quelques algorithmes fondamentaux choisis à la fois pour leur importance et pour leurs implications sur une structure de machine.

Ce choix est bien entendu arbitraire et non exhaustif ; pour chaque problème, le principe d'un algorithme est décrit, puis nous montrons qu'il peut être parallélisé ou traduit en termes de propagation.

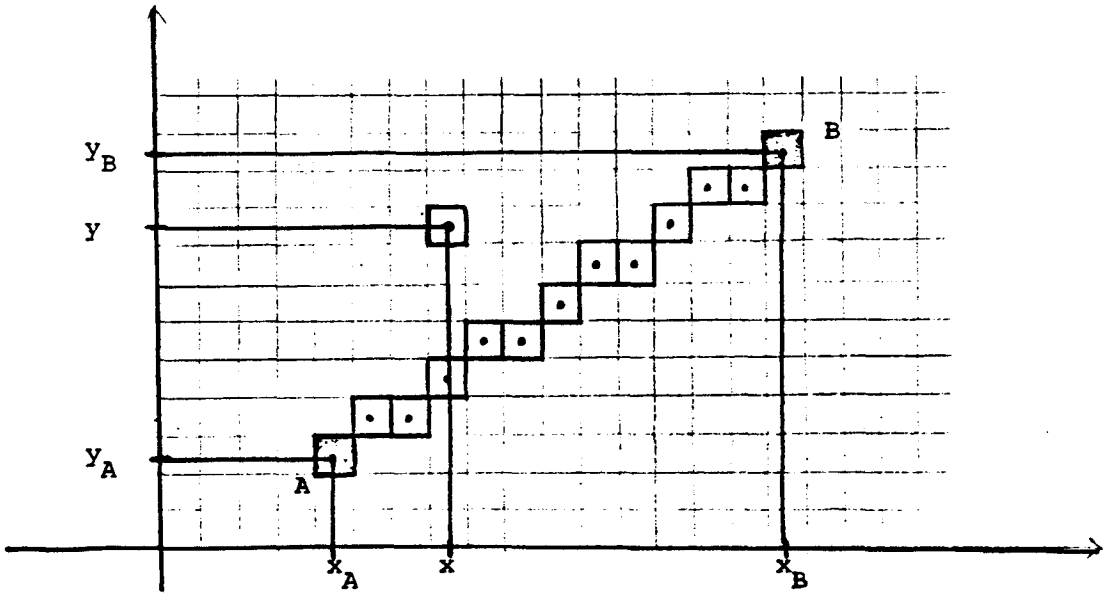
Nous montrerons également que les algorithmes choisis, et les architectures qui en résultent, s'appliquent également à la synthèse et au traitement.

III.2.1 - TRACE DE SEGMENT

Il s'agit là de l'une des activités de base du tracé d'un dessin ou d'une image. Cet algorithme peut également être utilisé pour toute interpolation linéaire entre deux valeurs.

III.2.1.1 - Le problème. Un algorithme séquentiel

Il s'agit de déterminer quels sont les meilleurs points d'une grille sur le parcours entre une origine A et une extrémité B.



Divers algorithmes résolvent plus ou moins correctement le problème. Nous retiendrons ici celui de Bresenham, repris par Lucas ([LUC 81]) que l'on peut formuler de la façon suivante :

Soit E une évaluation de l'erreur verticale entre un point idéal et le point étudié, et posons

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

Si nous supposons que $\Delta x \geq \Delta y \geq 0$, ce qui ne restreint pas la généralité du problème, nous obtenons :

```
   E = Δx ; y = yA ;
pour x = xA à xB
   faire  E = E + 2 * Δy
          si E > 2 * Δx alors faire y = y+1 ;
                                    E = E-2 * Δx ;
                               fait
          fsi
          écrire (x, y)
   fait ;
```

Cet algorithme évalue une erreur et détermine à partir de celle-ci quel est le meilleur point suivant. Dans le cas décrit où $\Delta x \geq \Delta y \geq 0$, il fournit un point pour chaque abscisse. Il peut être étendu aux cas où Δx et Δy sont quelconques. On trouvera dans [DUR 82] d'autres algorithmes d'interpolation et leur évaluation.

III.2.1.2 - Algorithme à propagation

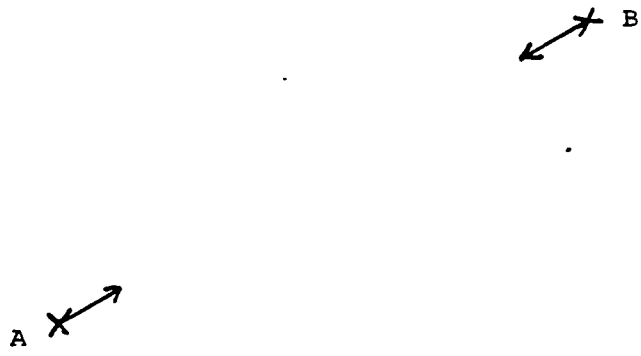
La boucle principale de l'algorithme ci-dessus indique de manière évidente une solution à propagation. Remarquons que l'on peut :

(1) démarrer le pipe-line de A et de B, divisant ainsi par 2 le temps de propagation

(2) pour chaque x, y déterminé, diffuser l'erreur E aux cellules voisines restantes, de manière à pouvoir réaliser un anti-aliasage partiel ; l'erreur verticale réelle étant $\varepsilon = E/(2\Delta x)$* en unités y

(3) cette propagation peut se faire dans toutes les directions et pas seulement dans le premier octant.

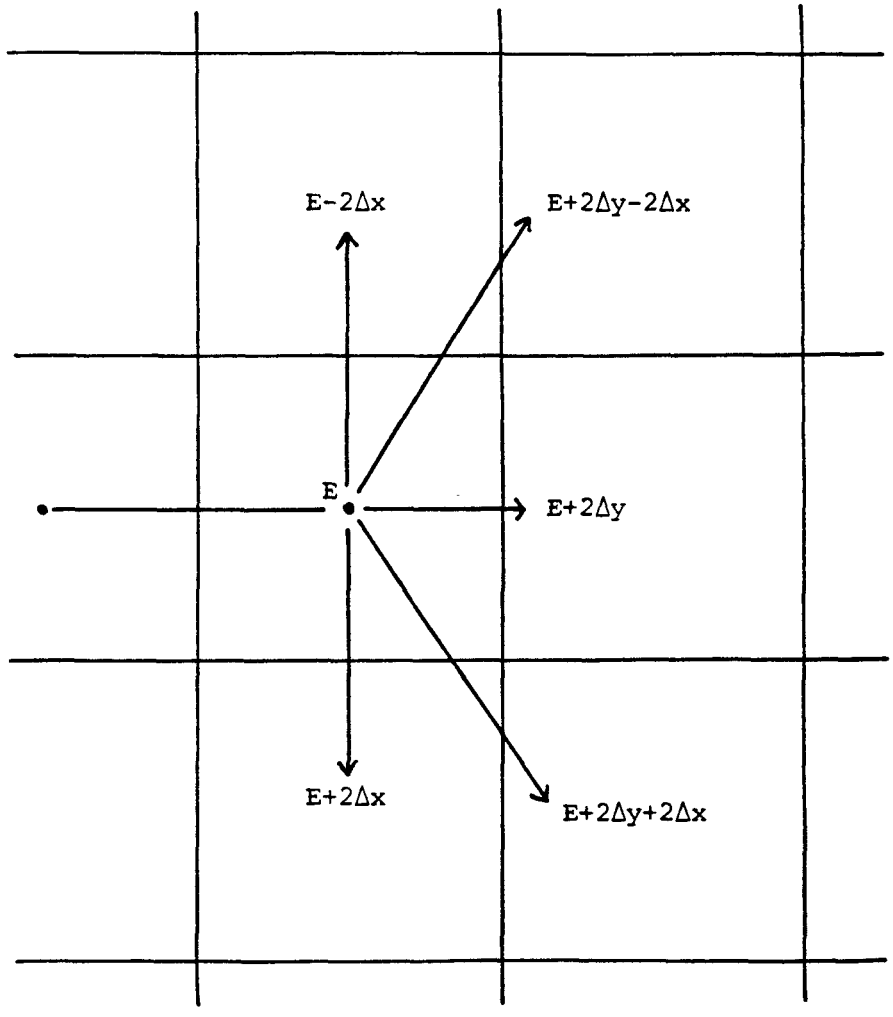
* *l'erreur réelle est* $d = \dfrac{d_v d_h}{\sqrt{d_v^2 + d_h^2}}$



(1)

---

(2) et (3)



III.2.1.3 - Algorithme parallèle

On peut enfin imaginer une version parallèle de cet algorithme, en remarquant que l'erreur verticale ε en un point quelconque (x, y) est

$$\varepsilon = (y-y_A) - (x-x_A) * \frac{\Delta y}{\Delta x}$$

On peut donc évaluer en tout point cette expression et la comparer à un seuil fixé (1/2 par exemple). La distance ainsi calculée permet également de traiter le problème de l'antialisage.

III.2.2 - REMPLISSAGE

III.2.2.1 - Le Problème

Une fois le contour d'un objet tracé, il s'agit de remplir ce contour, avec une valeur que nous supposerons ici constante (couleur, numéro de facette, etc ...). Ce problème se ramène à celui de trouver l'intérieur d'un polygone, et l'on connaît des algorithmes séquentiels permettant de le résoudre ([CEA 82]).

III.2.2.2 - Algorithme parallèle ([FUP 81])

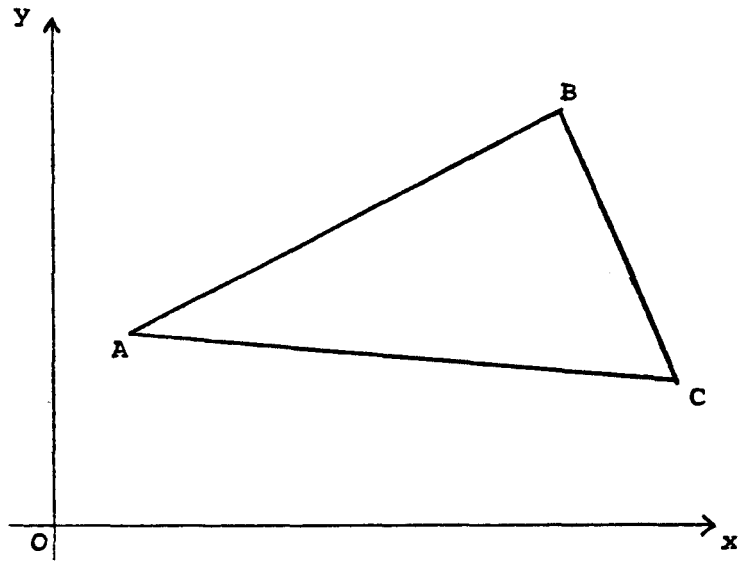
L'algorithme proposé est parallèle en ce sens que chaque point se détermine par rapport à chaque côté du polygone. Il ne fonctionne que pour des polygones convexes non troués.

Le principe est de déterminer pour tout (x, y) s'il se trouve par rapport au côté traité

- du côté du polygone
- de l'autre côté.

Cela se fait par calcul du signe de la puissance du point par rapport au côté, et comparaison avec un signe de référence. Un point appartient donc à l'intérieur du polygone s'il est du bon côté par rapport à chacun des côtés de ce polygone.

Exemple :



Les droites supports des côtés ont pour équations :

$$(AB) \qquad f_1 = a_1x + b_1y + c_1 = 0$$
$$(BC) \qquad f_2 = a_2x + b_2y + c_2 = 0$$
$$(CA) \qquad f_3 = a_3x + b_3y + c_3 = 0$$

En l'origine, on a $\begin{cases} f_1 = c_1 \\ f_2 = c_2 \\ f_3 = c_3 \end{cases}$

Si l'on note + lorsqu'un point est du bon côté d'un côté on doit avoir

$c_1 > 0$ (0 est du bon côté de AB)
$c_2 > 0$ (idem)
$c_3 < 0$ (0 est du mauvais côté !)

Une fois ce choix (arbitraire) de signe effectué dans les équations des droites supports, l'intérieur du polygone est déterminé par le système d'inéquations :

$$f_1 > 0$$
$$\text{et } f_2 > 0$$
$$\text{et } f_3 < 0$$

Remarque

Il s'agit donc d'effectuer dans chaque cellule des calculs du type

$$F = Ax + By + C,$$

expression que l'on sait répartir sous forme d'additions successives (arbre de Wallace) (voir machine de Fuchs, dans [FUP 81]).

III.2.2.3 - *Algorithme à propagation*

Cet algorithme, valable quel que soit le polygone, i.e. non convexe ou troué en particulier, est dérivé de l'algorithme de Conway (jeu de la vie) ; il consiste

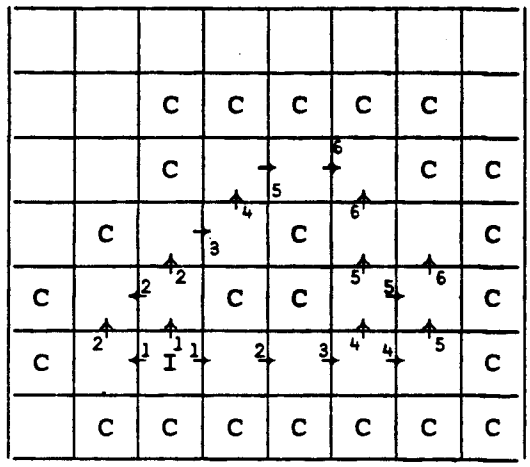
(1) à déterminer un (ou plusieurs) point(s) intérieur(s)

(2) à propager cette information d'intérieur chez les cellules voisines jusqu'à ce que l'on rencontre un obstacle (contour)

C

Exemple

(4 voisins)



Cet algorithme, très simple, nécessiterait en fait une cellule logique élémentaire présentée en annexe à ce chapitre. On peut évaluer son temps d'exécution comme étant la longueur du chemin conduisant du point initial au point le plus éloigné.

De plus il fonctionne quelle que soit la nature des contours, sous réserve de vérifier la continuité de ce contour, et de déterminer un ou plusieurs points intérieurs.

III.2.3 - SURFACES VISIBLES

Compte tenu de la nature cellulaire de la classe d'architectures envisagée, la technique dite du "tampon en z" ([CAT 79]) paraît la plus appropriée : Elle consiste à comparer en tout point la nouvelle valeur de z que l'on veut introduire (z_c) avec l'ancienne présente (z_r) puis ; si z_c est supérieur à z_r, alors on conserve z_r ; sinon on prend dorénavant z_c.

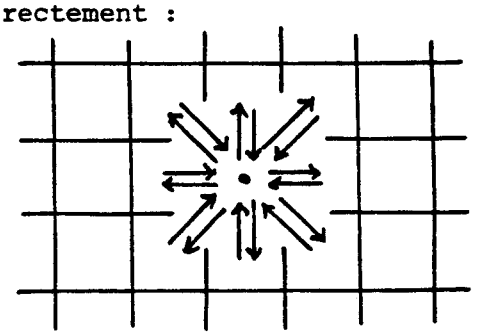
Le fonctionnement est donc totalement parallèle (SIMD), mais chaque polygone est traité séquentiellement.

III.2.4 - LISSAGE - FILTRAGE

Ces techniques font partie d'une classe de problèmes faisant intervenir plusieurs cellules pour l'évaluation d'une seule.

III.2.4.1 - Solution parallèle

Il existe une solution totalement parallèle si le réseau est connecté de manière à ce qu'une cellule puisse accèder aux informations des voisines imposées par l'algorithme. Ainsi, sur un réseau à 8 voisins, un filtrage 3×3 est possible directement :



Mais cela n'est plus possible

- si le nombre de voisins augmente
- si le réseau est moins connecté.

On est alors conduit à séquentialiser les accès (i.e. propager les données) et les calculs.

III.2.4.2 - Solution à propagation de données

Dans le cas général d'un filtre n×n (n impair), et dans l'hypothèse d'un réseau à 4 voisins, il faut réaliser la propagation de n^2-1 valeurs vers la cellule centrale. Un procédé de propagation sera étudié ultérieurement, qui garantit une propagation totale et unique.

III.3 - ARCHITECTURE THEORIQUE A UN NIVEAU

III.3.1 - PRINCIPES GENERAUX

III.3.1.1 - Principes

Le tableau suivant résume l'adéquation des modes de fonctionnement proposés en § III.2 par rapport aux algorithmes que nous venons d'étudier.

	SIMD	MIMD	macro SIMD	propagation
tracé de segment parallèle	X			
"séquentiel"			X	X (données et/ou contrôle)
remplissage parallèle	X			
"séquentiel"			X	X (données et/ou contrôle)
surfaces visibles	X			
lissage	X			X(données)

Un SIMD vrai n'étant qu'un cas particulier de macro SIMD, nous faisons les hypothèses suivantes :

. le fonctionnement sera du type macro-SIMD au niveau du contrôle (éventuellement micro-SIMD), i.e. que chaque cellule exécute le même processus (programme ou microprogramme)

. le fonctionnement sera du type à propagation pour les données.

Ces hypothèses sont accompagnées des choix d'implémentation suivants :

1) La cellule est un automate.

2) L'espace cellulaire associé est un pavage carré du plan.

3) Le voisinage sera de 4 cellules, pour des raisons liées à la technologie, à la logique des algorithmes et à la nature du balayage du visuel supposé.

4) Chaque cellule peut, à partir de ses valeurs et des valeurs transmises par les voisines, évaluer de nouvelles valeurs et les transmettre à ses voisines.

De manière à simplifier la description ultérieure d'exemples, et sans préjuger d'une implémentation réelle, nous considérerons qu'une cellule peut se trouver dans trois états :

* état repos : la cellule ne fait rien

* état actif : la cellule travaille et fournit éventuellement des valeurs aux voisines

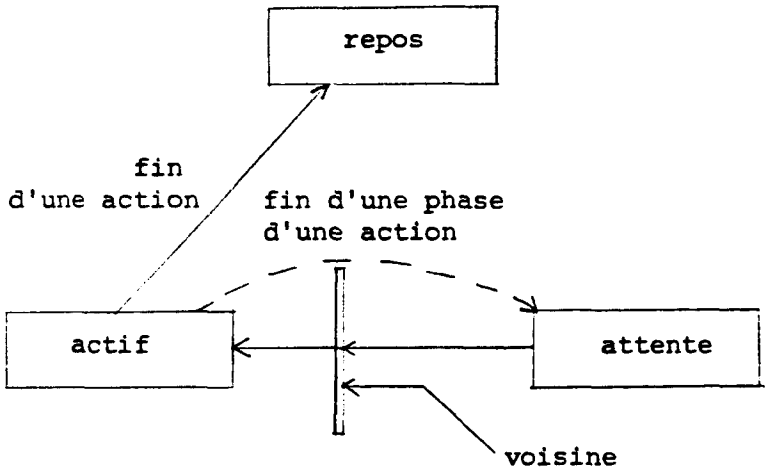
* état attente : la cellule peut être activée sur réception de valeurs des voisines.

De manière à pouvoir contrôler le déroulement des algorithmes, ces états peuvent être forcés de l'extérieur sélectivement pour certaines cellules.

Notons ici cependant que notre hypothèse de propagation des données seules dans le réseau sous-entend qu'une partie du contrôle aussi est propagé, car bien entendu ces données influent sur le contrôle des cellules atteintes.

III.3.1.2 - Diagramme d'états

En négligeant le forçage pour le contrôle extérieur, la succession des états peut être représentée par le diagramme suivant



Supposons que nous souhaitions réaliser l'action A sur un domaine D ; il faut alors procéder de la façon suivante

* toutes les cellules étant initialement au repos, on met en attente sur A les cellules c_i du domaine D

* on rend active sur A une ou plusieurs cellules de D où A est directement réalisable, éventuellement toutes

* des fronts d'ondes d'activité se propagent : chaque cellule activée exécute l'action A, puis se met au repos, après avoir éventuellement activé une ou plusieurs voisines

* l'action A est terminée lorsqu'il n'y a plus de cellule active (une solution à ce problème délicat sera proposée ultérieurement).

En fait, suivant l'algorithme, on aura en général deux types de fonctionnement assez distincts :

1) SIMD vrai, avec ou sans propagation de données, celles-ci n'influant pas sur l'état des cellules, toutes actives ; le cas typique est celui du filtrage.

2) macro-SIMD, avec propagation de données activant au passage les cellules en attente ; le cas typique est le tracé de segment (par propagation) ou le remplissage (par "jeu de la vie").

III.3.2 - MOYENS DE CALCUL, DE CONTROLE ET DE MEMORISATION

III.3.2.1 - Moyens de calcul

Le tableau ci-dessous indique les besoins en calcul et en mémoire des différents algorithmes étudiés, dans une certaine implémentation.

algorithme	opérateurs (sur entiers)	variables temporaires	valeurs à mémoriser
segment	$\geq$ + -	1	3 + 1 booléen
remplissage	$\cap$ $\cup$	1 (booléen)	3 booléens
z-buffer	$\geq$	0	2
filtrage	$\pm$ *	1	5

Ce tableau montre clairement qu'une unité arithmétique et logique simple associée à un minimum de 6 registres de travail suffit à satisfaire les besoins en calcul.

III.3.2.2 - Moyens de mémorisation et de contrôle

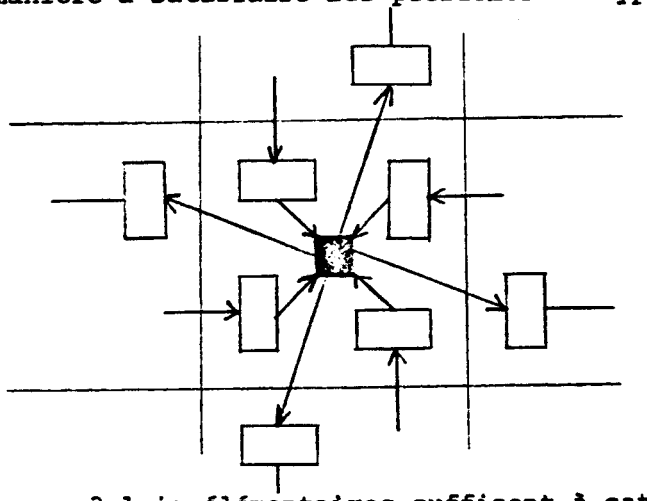
Il est nécessaire que chaque cellule puisse mémoriser le processus à exécuter ; diverses implémentations sont possibles (RAM + chargement, PROM ...) et le choix ne nous paraît pas être essentiel. De même le problème du contrôle par l'extérieur a de nombreuses solutions sur lesquelles nous reviendrons.

III.3.3 - COMMUNICATIONS

III.3.3.1 - Communications des données dans le réseau

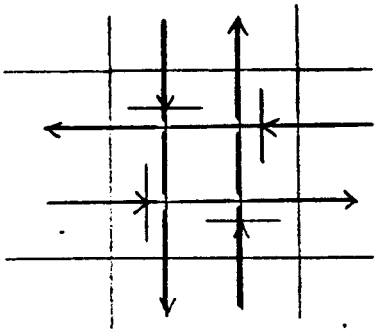
On choisit de pouvoir communiquer de manière bidirectionnelle avec les

4 voisins, de manière à satisfaire les problèmes du type "filtrage".

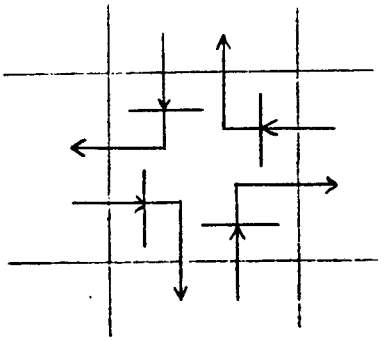


Nous constatons que 3 lois élémentaires suffisent à satisfaire les algorithmes étudiés (et beaucoup d'autres).

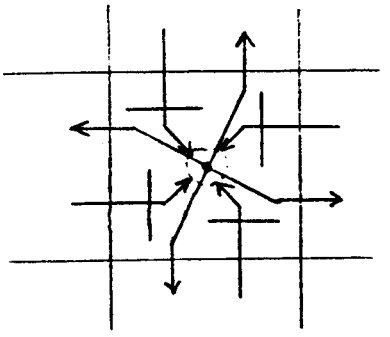
(1) en ligne droite



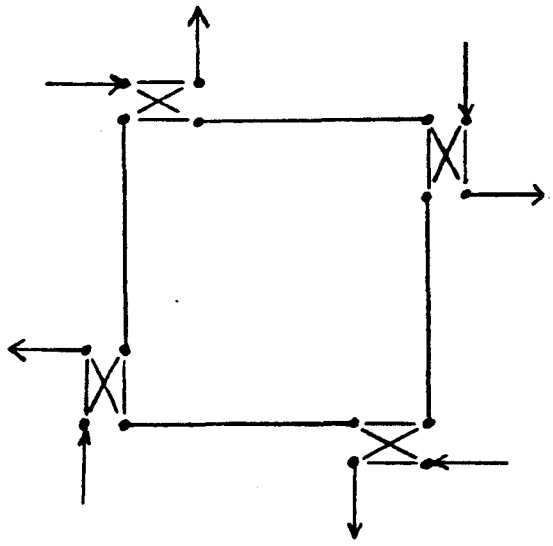
(2) à angle droit



(3) diffusion



Nous choisissons a priori un fonctionnement **synchrone** de ces communications, une solution asynchrone étant moins simple. Dans ce cas l'existence même des 4 registres de communication est discutable.
La réalisation matérielle de ces communications pourrait être aisément assurée par l'utilisation de cellules de commutation élémentaires (Banyan) par exemple de la manière suivante (pour (1) et (2)).



Dans le cas où une réalisation synchrone serait impossible, il faudrait bien entendu mettre en oeuvre un protocole de communication adéquat.

Notons de plus que si dans les cas (1) et (2) seules deux cellules interviennent (la cellule centrale ne servant qu'à transmettre), dans le cas (3), trois cellules sont en jeu et le processus de synchronisation est plus complexe.

III.3.3.2 - *Moyens de communication avec l'extérieur*

Les communications avec l'extérieur sont nécessaires dans deux cas :

- diffusion générale de commandes et données initiales
- récupération des résultats.

1) Diffusion de commandes et de données initiales

On cherche à effectuer une diffusion générale d'une commande ou d'une donnée ; en supposant que chaque cellule puisse reconnaître si la commande la concerne, il n'est pas alors utile de traiter le problème de la diffusion sélective.

Deux procédés principaux sont utilisables :

* La propagation à l'aide des moyens de communication existants dans le réseau, à partir des bords, ou de points particuliers. On peut montrer que

pour un réseau toroïdal n×n à p points de propagation régulièrement répartis, le temps de diffusion T est

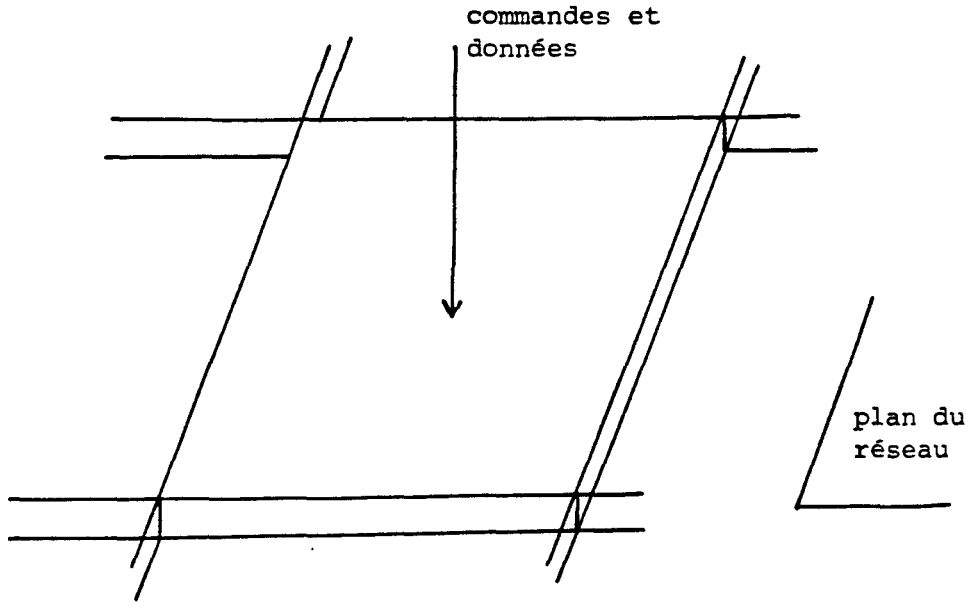
$$T = \frac{n\tau}{p}$$

si τ est le temps de cycle élémentaire de propagation.

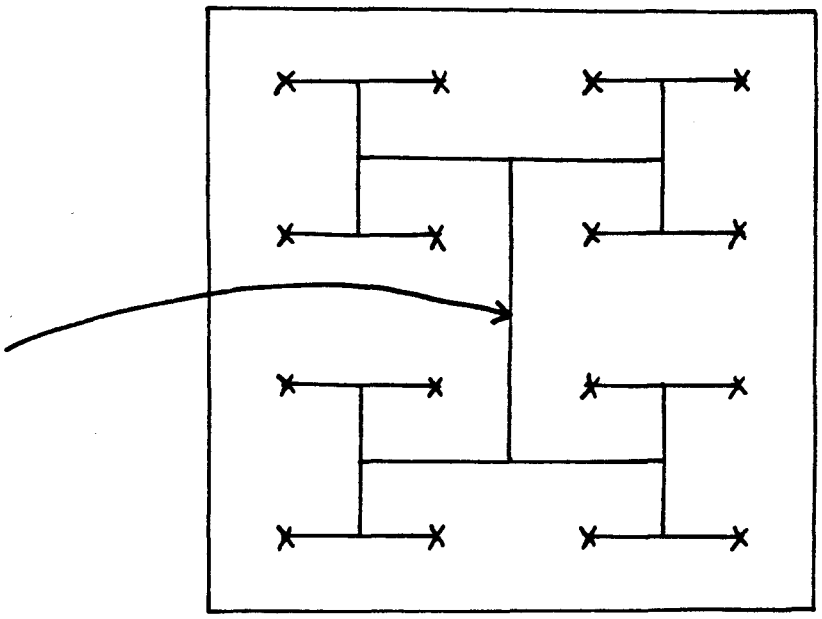
L'inconvénient de ce procédé est de requérir le moyen de communication interne du réseau, et donc de bloquer celui-ci.

* La diffusion simultanée

- soit à l'aide d'un câblage supplémentaire, chaque cellule disposant de k fils de réception générale, avec les problèmes de connections que cela suppose.
- soit à l'aide d'une troisième dimension, pas obligatoiremement câblée, mais par exemple optique, ou électronique permettant la réception simultanée de commandes et données.



Nous pouvons citer également dans cette classe le procédé très astucieux de propagation arborescente, souvent utilisé pour propager des signaux dans les VLSI, qui garantit que le parcours effectué par le signal diffusé est le même pour toutes les cellules. Ce procédé assure donc de fait le synchronisme des opérations, ([FIK 83]).



En ce qui concerne la structure des messages diffusés, nous proposons que cette réception se fasse associativement à l'aide du format suivant :

X_D	X_F	Y_D	Y_F	data/cmd

qui permet de diffuser en parallèle sur une zone rectangulaire (D, F).

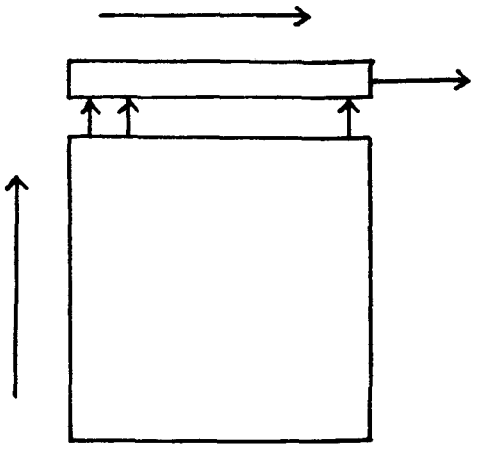
La cellule déroule alors l'algorithme :

si $(X_D \leq X \leq X_F)$ et $(Y_D \leq Y \leq Y_F)$
    alors cmd → CMD
           data → R(x)
    } suivant le cas
finsi

2) Récupérations des résultats

Compte tenu du balayage tramé du dispositif de visualisation, il s'agit ici essentiellement de récupérer ligne à ligne le contenu du réseau. Deux méthodes essentielles sont envisageables :

- l'une, très lourde, consistant à superposer un système d'adressage et un bus de sortie du réseau
- l'autre, plus logique dans notre cas, consiste à vider le réseau par un bord, à l'aide de décalages successifs, en utilisant le moyen de communication fourni par le réseau.

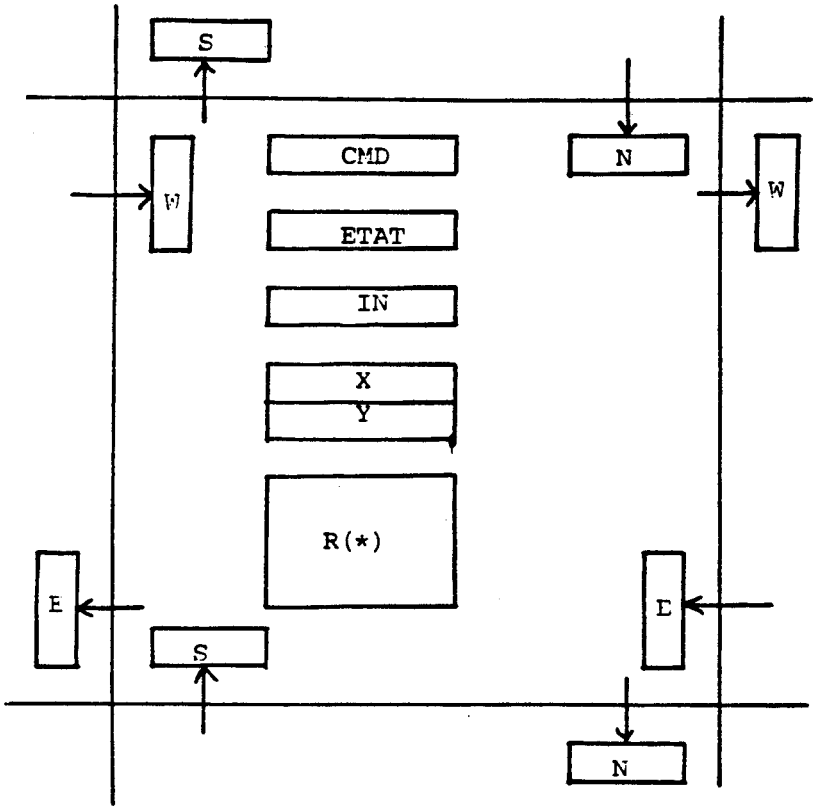


III.3.4 - DESCRIPTION D'UNE ARCHITECTURE POSSIBLE

III.3.4.1 - La Cellule

Une cellule comporte

- 4 registres de communication recevant les informations venant des voisines
- une bascule IN indiquant si l'un au moins des registres de communication contient une valeur
- un registre CMD indiquant la commande à effectuer par la cellule (il s'agit de macro-commande dans l'hypothèse envisagée)
- un registre ETAT indiquant l'état de la cellule
  - Ø inactive
  - 1 en attente, i.e. activable par IN ≠ 0
  - 2 active, i.e. exécutant CMD
- deux registres X, Y indiquant les coordonnées de la cellule dans le réseau nécessaires pour le contrôle sélectif et certains algorithmes
- des registres de travail (6)
- une ALU, dont la complexité reste à définir
- un moyen de **réception** de données/commandes.



La cellule fonction de la manière suivante :

```
cas      ETAT
  ;  /* cellule inerte ; état repos */
  ;  si IN = vrai alors cas  CMD ;
                              /* IN active la cellule */
                         fincas
     finsi
  ;        cas  CMD ;
                /* cellule active en elle-même */
           fincas
  ETAT = 0 ; /* repos */
fincas
```

III.3.4.2 - Caractérisation du réseau inerte

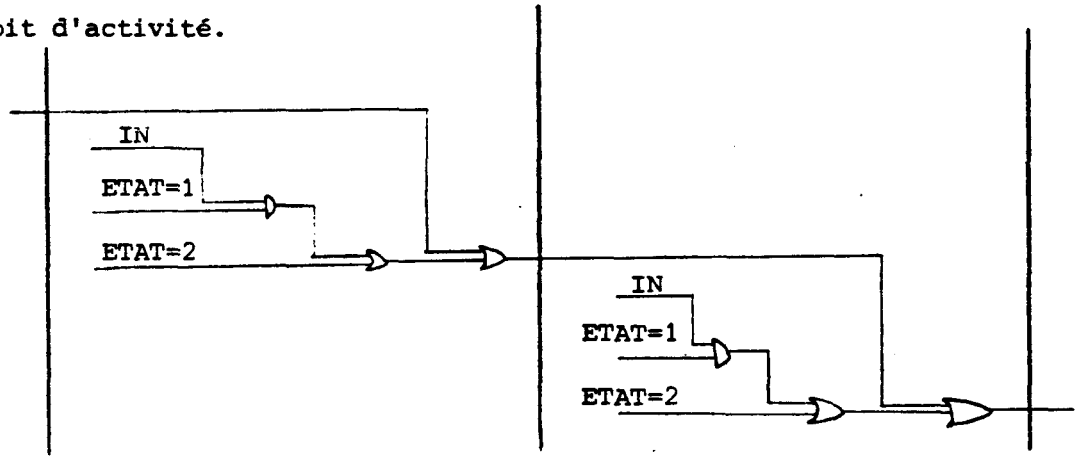
Le réseau est inerte si et seulement si

$$\forall\ x, y \begin{cases} (C(x, y).IN = 0 \text{ et } ETAT = 1) \\ \text{ou} \qquad\qquad ETAT = 0 \end{cases}$$

donc il est actif si

$$\exists\ x, y \text{ tq} \begin{cases} IN = 1 \text{ et } ETAT = 1 \\ \text{ou } ETAT = 2 \end{cases}$$

Ceci est détectable de l'extérieur en propageant de cellule en cellule 1 bit d'activité.



Remarquons que ce schéma n'est valable que si le temps de propagation de ce bit est inférieur au temps élémentaire de propagation de données dans le réseau ; sinon il y a risque de malfonctionnement.
Ce bit pourrait par exemple être transmis horizontalement, la colonne extrême devant être analysée pour obtenir l'indication finale d'activité.

$$A = \bigcup_{\text{lignes}} A_i$$

III.3.5 - PERFORMANCES

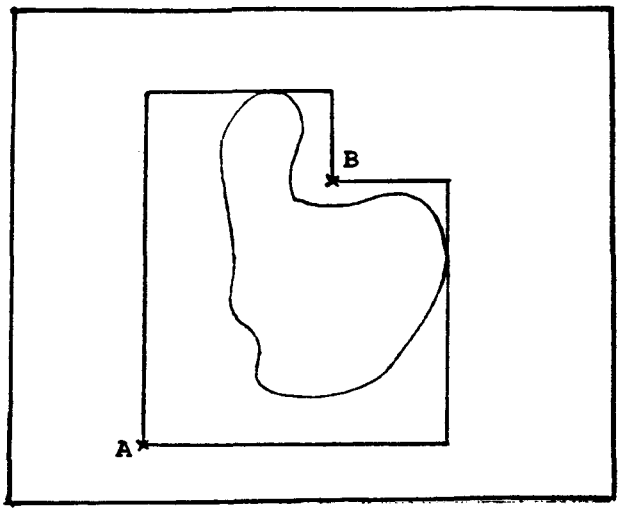
III.3.5.1 - Notion de distance, de taille

Lorsqu'un algorithme se prête à un fonctionnement en propagation, il est clair que la durée de son exécution dépend

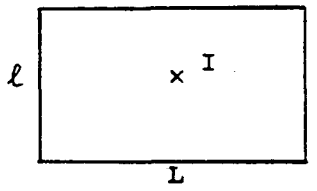
1) du temps de cycle de chaque cellule τ

2) du nombre de cellules à activer successivement pour atteindre tous les points.

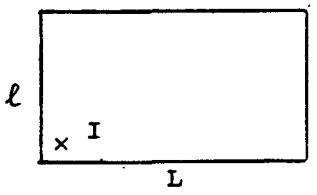
Nous appelerons donc **"distance"** entre deux cellules, la longueur du parcours (horizontal ou vertical dans le cas d'un réseau à 4 voisins) le plus court reliant l'une à l'autre, compte tenu de détours éventuels (voir figure)



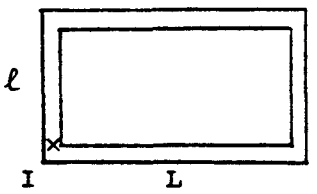
Nous en déduisons la notion de **"taille"** d'un objet, notion importante pour tous les algorithmes liés au remplissage :
la taille d'un objet O connexe (par rapport à un point intérieur I) est la distance maximale à parcourir pour atteindre à partir de I tous les points intérieurs à O.
Exemples:



$t = (\ell+L)/2$



$t = \ell+L$



$t = \ell+L$

Il est clair que la taille ainsi définie dépend :

- du point intérieur choisi
- de la forme des contours de l'objet.

Elle est calculable dans des cas simples (tels que ceux présentés) mais pas en général. On peut cependant fréquemment la borner par un majorant (et un minorant également).

Elle a de plus les propriétés suivantes :

. la taille est invariante par translation

. une homothethie h transforme une taille t en taille h*t

. une taille t est généralement modifiée par rotation en une nouvelle taille t' majorable par $t\sqrt{2}$ et minorable par $t/\sqrt{2}$.

. si $I \in O_1 \cap O_2$ alors $t(O_1 \cup O_2) \leq \sup (t(O_1), t(O_2))$

III.3.5.2 - Temps d'exécution d'un algorithme

En ce qui concerne les algorithmes parallèles (filtrage, z-buffer, etc ...) ce temps est égal au temps τ d'exécution de l'algorithme par une cellule

$$\theta = \tau$$

Pour ce qui est des algorithmes à propagation, dans le cas d'une seule source de propagation, ce temps est

$$\theta = t \times \tau$$

(t étant la taille du domaine, relative au point d'injection).

Dans le cas de plusieurs sources de propagation, il est difficile d'estimer le temps t ; on a une borne supérieure évidente qui est

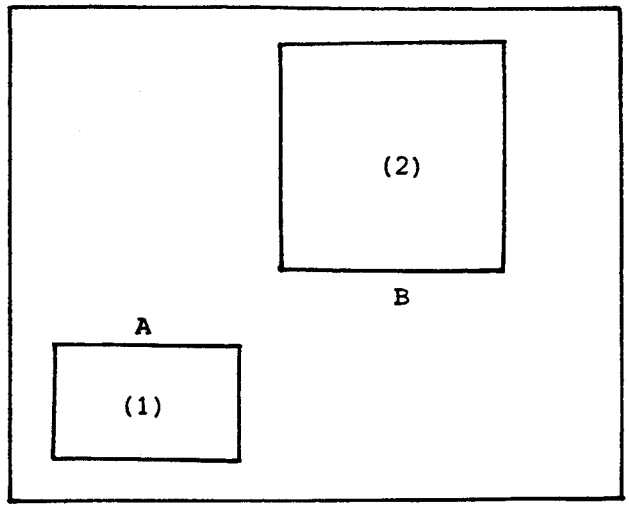
$$\theta' \leq \inf (t_1 \tau, t_2 \tau) = \tau * \inf (t_1, t_2)$$

Des exemples simples montrent que si l'on a n sources de propagations pour un objet "bien fait", on a probablement

$$\theta' \simeq \theta/n$$

III.3.5.3 - Opérations en parallèle

Considérons deux zones A et B du réseau, indépendantes. Nous pouvons exécuter sur A une opération 1 et sur B une opération 2 (exemple)



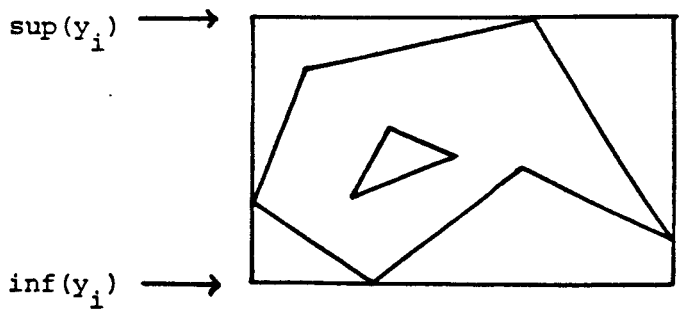
Nous obtenons ainsi un parallèlisme actions et/ou objets qui aboutit à une architecture MIMD. L'étude menée au chapitre I sur ces cas de parallélisme simple montre que l'on sait découper un ensemble objets X actions en sous-ensembles exécutables en parallèle.

On a généralement à effectuer des opérations sur des domaines correspondants à des objets ; compte tenu de la nature carrée du réseau choisi, une solution simple garantissant l'indépendance de deux domaines consiste à

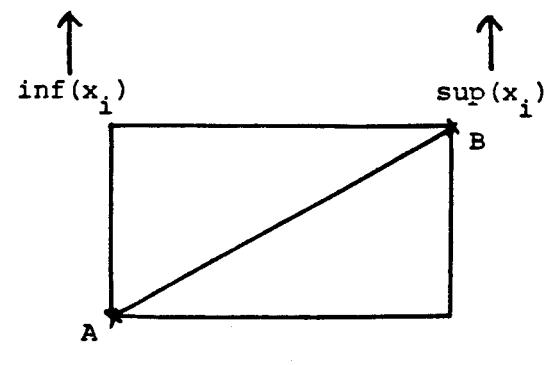
1) étendre chaque domaine à un rectangle englobant définit par (min (x), min (y), max (x), max (y)).

2) comparer ces rectangles entre eux.

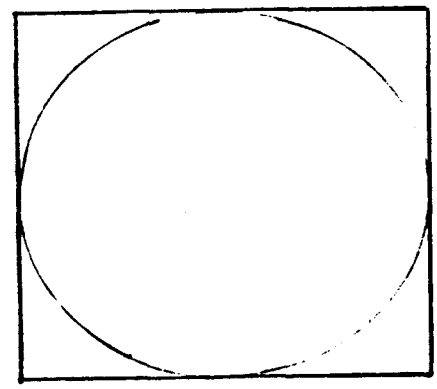
<u>polygone</u>



<u>vecteur</u>



cercle/ellipse



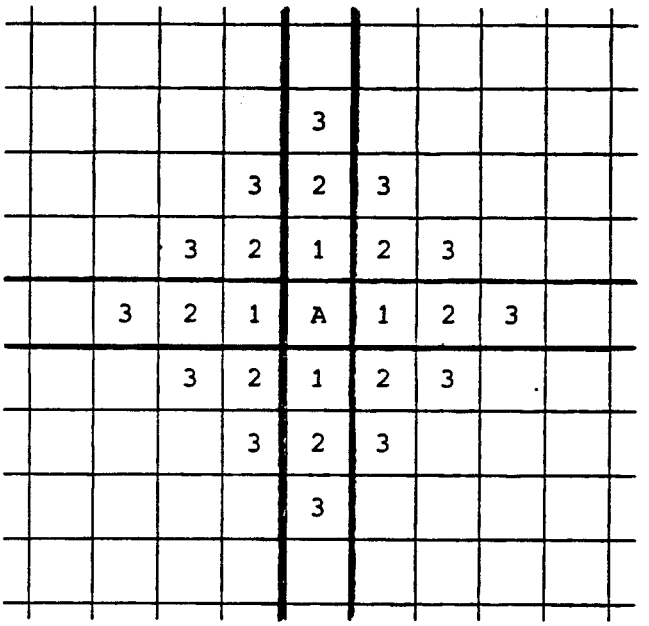
Pour un ensemble domaines × actions, on peut donc construire une table de "disjonction", que l'on pourra segmenter le mieux possible (voir annexe 1).

III.4 - IMPLEMENTATION D'ALGORITHMES CELLULARISES

Nous donnons dans cette partie quelques exemples d'implémentation d'algorithmes de synthèse, traitement et manipulation d'images. Dans chaque cas on indiquera quelques points théoriques, puis l'implémentation au niveau contrôle et au niveau exécution cellulaire.

Deux remarques préliminaires s'imposent :

1 - Etant donné un algorithme à propagation initialisé à partir d'une cellule A, l'activité se propage en général suivant un front d'onde de forme { carrée à partir de A / losange



2 - Il est souvent nécessaire qu'une cellule connaisse sa position (x, y) dans le réseau. Nous verrons que cela résulte immédiatement de l'évaluation de deux fonctions du type f(x, y) simples.

III.4.1 - ALGORITHMES DE SYNTHESE D'IMAGES

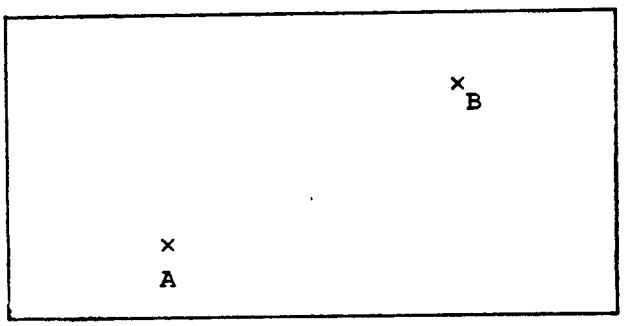
De manière générale ([MAR 82]), on cherche à effectuer la synthèse de l'identité, de la morphologie, de la géométrie, de l'aspect et de l'éclairement. Nous avons choisi ici quelques algorithmes fondamentaux, les plus généraux possibles. Les trois premiers (tracé de segment et de cercle/disque, remplissage uniforme) concernent la morphologie et la géométrie, l'évaluation d'une fonction f(x, y) du second degré est utile en morphologie (tracé de coniques) et en aspect/éclairement (calculs d'ombrages).

Il nous a paru utilse de donner quelques indications sur un algorithme de surfaces visibles (z-buffer) et sur un algorithme de découpage ; de même est proposée une approche du problème du lissage. Les problèmes de filtrage sont reportés dans la partie III42 (traitement d'images) ; les problèmes toujours sous-jacents d'échantillonage et de quantificaiton seront traités dans la mesure du possible au fur et à mesure .

III.4.1.1 - Tracé de segment

1. Aspect théorique

Nous étendrons ici quelque peu le problème du tracé de segment déjà décrit en III.2.1 : en effet notre réseau permet de déterminer, sans perte de performance, une distance d'un point quelconque au segment idéal. Partant des extrémités A et B, cette erreur est calculable sur le domaine D.



soit $\Delta y = y_B - y_A$

$\Delta x = x_B - x_A$

on a $y = \frac{\Delta y}{\Delta x} x + b$ sur la droite (avec $b = y_A - \frac{\Delta y}{\Delta x} x_A$).

Soit $\varepsilon(x, y)$ l'erreur verticale d'un point (x, y) par rapport à la droite réelle.

$$(x, y) = y - \frac{\Delta y}{\Delta x} x - b$$

On en déduit immédiatement

$$\begin{cases} \varepsilon(x+1, y) = \varepsilon(x, y) - \frac{\Delta y}{\Delta x} \\ \varepsilon(x-1, y) = \varepsilon(x, y) + \frac{\Delta y}{\Delta x} \\ \varepsilon(x, y+1) = \varepsilon(x, y) + 1 \\ \varepsilon(x, y-1) = \varepsilon(x, y) - 1 \end{cases}$$

En partant de A et B, où l'erreur est nulle, on peut calculer cette "distance" verticale en tout point (x, y) du domaine D. On trouvera en Annexe une utilisation de cette distance pour traiter l'antialiassage des segments. Si l'on cherche simplement à tracer un segment brut, il suffit d'avoir un critère sur ε permettant de dire si l'on a un "bon" point ou non ; ce critère est le suivant

$$\text{pour } |\Delta x| \geq |\Delta y| \qquad |\varepsilon| < \frac{1}{2}$$

$$\text{pour } |\Delta x| \leq |\Delta y| \qquad |\varepsilon| < \left|\frac{\Delta y}{\Delta x}\right| * \frac{1}{2}$$

Remarquons pour terminer que si A et B ne sont pas exactement sur le segment, il suffit d'initialiser ε_A et ε_B à des valeurs précalculées.

L'implémentation effectuera le calcul de $E = \varepsilon * 2\Delta x$.

2. Contrôle

Il s'effectue en deux temps :

1) Initialisation globale du domaine D de travail avec les valeurs suivantes

- ETAT = 1 /* attente */
- R(2) = 2 * Δx (incrément en y)
- R(3) = 2 * Δy (incrément en x)
- R(5) = seuil, i.e. |Δx| ou |Δy| suivant octant
- R(4) servira au calcul de l'erreur E
- R(0) contiendra 1 sur le segment

2) Activation de points remarquables, ici A et B, par forçage de ETAT à 2.
Remarquons l'existence d'autres points remarquables :
En effet, si Δy et Δx ne sont pas premiers entre eux, soit p leur PGCD ; alors

$$\left(x_A + k * \frac{\Delta x}{p} , y_A + k * \frac{\Delta y}{p}\right) \text{ sont des points exacts du segment.}$$

3. Exécution au niveau cellulaire

Chaque cellule activée exécute l'algorithme suivant :

```
SEGMENT : cas  IN ; /* différencie la phase initiale de la suite */
          |R(4) = 0 ; /* E = 0 en A et B */
          |R(4) = N or E or S or W ; /* cas général */
          fincas
      NORD.S = R(4) + R(2) ; /*
      SUD.N  = R(4) - R(2) ; propagation de
      EST.N  = R(4) + R(3) ;       l'erreur
      WEST.E = R(4) - R(3) ;                */
      si |R(4)| ≤ R(5)
         |alors R(0) = 1 ; /* résultat */
         |sinon R(0) = 0 ;
      fsi
      /* traitement éventuel d'anti-aliassage */
      ETAT = Ø ; /* arrêt */
```

4. Performances

Si τ est le temps de cycle du réseau, le temps d'évaluation de l'erreur sur le domaine D est

$$T = \frac{\tau}{2} * \frac{\Delta x+\Delta y}{p}$$

(en supposant la détection d'inactivité instantanée ou de durée négligeable).

III.4.1.2 - Tracé de cercle/disque

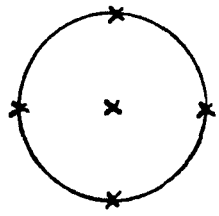
1. Aspect théorique

Dans ce cas également, de nombreux auteurs (dont l'inévitable (!) Bresenham) se sont donné la peine d'étudier le problème. Il s'agit le plus souvent de trouver la meilleure approximation d'un cercle de centre 0 et de rayon R. En suivant la même démarche que précédemment, nous proposons une solution parallèle à propagation qui permet de déterminer un disque ou le cercle frontière à partir d'un calcul de distance, qui est ici la puissance d'un point par rapport au cercle, supposé centré en (0, 0).

Soit $\varepsilon(x, y) = x^2 + y^2 - R^2$, puissance d'un point (x, y) par rapport au cercle. Nous noterons que

1) $\varepsilon = 0$ sur le cercle idéal
2) $\varepsilon \leq 0$ sur le disque associé
3) $\sqrt{\varepsilon}$ est la distance effective du centre du pixel du cercle idéal (voir en Annexe le traitement de l'aliassage)

4) On connait 5 points remarquables qui sont le centre ($\varepsilon = -R^2$) et 4 points sur le cercle ($\varepsilon = 0$).



5) Dans le cas où l'on traiterait une sphère centrée en (0, 0, 0) ε représente $-z^2$ de la sphère.

Nous obtenons

$$\begin{cases} \varepsilon(x+1, y) = (x, y) + 2x+1 \\ \varepsilon(x-1, y) = (x, y) - 2x+1 \\ \varepsilon(x, y+1) = (x, y) + 2y+1 \\ \varepsilon(x, y-1) = (x, y) - 2y+1 \end{cases}$$

Pour pouvoir tracer le cercle ou limiter la sphère il faut pouvoir déterminer un seuil S. Ce seuil S dépend du critère de ressemblance que l'on cherche par rapport au cercle idéal ; la perception de continuité nous impose de choisir que la limite du cercle fera au moins un pixel d'épaisseur en x et en y. En première approche, il faut donc vérifier pour tout (x, y) proche du cercle:

$$\begin{cases} S \geq \frac{1}{2}|\ 2x+1| \\ S \geq \frac{1}{2}|-2x+1| \\ S \geq \frac{1}{2}|\ 2y+1| \\ S \geq \frac{1}{2}|-2y+1| \end{cases}$$

ce que l'on ramène à (dans le premier octant)

$$S \geq \frac{1}{2}\ (2x+1)$$

x variant de R à E ($\frac{R\sqrt{2}}{2}$), on obtient une valeur trop grande de seuil, toujours acceptable en choisissant

$$S = R+1$$

La simulation (voir Annexe) corrige en fait cette approche pessimiste par une valeur plus efficace

$$S = R-1$$

qui a comme seul inconvénient de donner un cercle légèrement différent de celui de Bresenham (mais qui a raison ?).

2. Contrôle

Il s'effectue en trois temps :

1) Initialisation du pavé de travail D avec les valeurs suivantes :

- ETAT = 1
- R(2) = X_c (abscisse du centre)
- R(3) = Y_c (ordonnée du centre)
- R(5) = R-1 (valeur du seuil)
- R(4) contiendra la valeur de ε
- R(0) = 1 sur le cercle [le disque]

2) Calcul des valeurs $2(x-x_c)$ et $2(y-y_c)$ par activation de tout le pavé D et exécution cellulaire de C_I.

3) Calcul effectif par initialisation de D à ETAT = 1 et activation des points remarquables choisis, puis exécution cellulaire de C_{II}.

3. Algorithmes cellulaires :

→ C_I permet d'évaluer les expressions $2(x-x_c)$ et $2(y-y_c)$
Chaque cellule calcule en parallèle

```
{ R(2) = 2 * (X-R(2))
{ R(3) = 2 * (Y-R(3))
```

→ C_{II} trace le cercle (le disque) proprement dit.

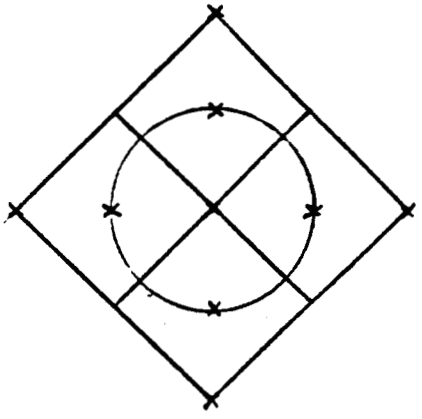
```
CERCLE : cas IN ;
         |R(4) = 0 ; /* -R² si on part du centre */
         |R(4) = N or E or S or W ;
         fincas
  NORD.S = R(4) + R(3) + 1 ;
  SUD.N  = R(4) - R(3) + 1 ;
  EST.W  = R(4) + R(2) + 1 ;
  WEST.E = R(4) - R(2) + 1 ;
 [si  R(4)   ≤ R(5) alors R(∅) = 1 ; /* cas disque */]
  si |R(4)| ≤ R(5) alors R(∅) = 1 ; /* cas cercle */
  ETAT = ∅
```

4. Performances

Si les sources de propagation sont les 4 points remarquables cités en § IV.1.3.1, le temps d'évaluation dans un domaine D minimal recouvrant le cercle est

$$T = R * \tau$$

L'ajout du centre n'apporte aucune amélioration



Par contre l'utilisation du centre seul est inefficace, car nous conduit à

$$T = R\sqrt{2}\ \tau$$

Notons ici encore l'utilisation de ε qui peut être faite pour traiter le problème du sous-échantillonage (voir Annexe générale 2).

III.4.1.3 - Remplissage

1. Aspect théorique

Déjà présenté en § III.2, ce problème a également de nombreuses solutions ; mais nous retiendrons bien évidemment celle qui consiste à propager une information d'intérieur jusqu'aux bords de l'objet. Cet algorithme permet de traiter des polygones quelconques (convexes ou non, troués ou non), et aussi des objets constitués de plusieurs polygones disjoints sous réserve que :
1) Le contour soit continu au sens du réseau, c'est à dire qu'on ne puisse pas le traverser sans s'en apercevoir
2) on connaisse un (ou plusieurs) points intérieurs.

2. Contrôle

Après inscription du contour, par exemple dans le registre R(Ø), il faut initialiser tout le domaine de travail à

ETAT = 1

puis activer le ou les points intérieurs. (i.e. ETAT = 2)

R(1) contiendra l'information d'intérieur.

3. Exécution au niveau cellulaire

```
REMPL :  R(1) = 1 ;
         si R(0) = 0  alors NORD.S = SUD.N = EST.N = EST.W = WEST.E = 1 ;
         ETAT = 0 ;
```

Nota : une cellule de contour est considérée comme intérieure, mais ne propage pas d'information d'intérieur.

Comme nous l'avons déjà signalé (§ III.2), cet algorithme très simple peut se réaliser à l'aide de quelques portes élémentaires (voir en Annexe à ce chapitre).

4. Performances

Elles sont liées au nombre de points intérieurs initiaux et à la taille du polygone.

III.4.1.4 - Evaluation d'une fonction F(x, y)

1. Fonction du second degré

Nous avons montré dans les cas particuliers du tracé de segment et du tracé de cercle que le réseau permet d'évaluer en tout point une fonction F(x, y) du type

$$F_1(x, y) = ax + by + c$$

ou $$F_2(x, y) = (x-x_o)^2 + (y-y_o)^2 + d$$

à l'aide de calculs d'incréments

$$\begin{cases} \Delta+x = F(x+1, y) - F(x, y) \\ \Delta-x = F(x-1, y) - F(x, y) \\ \Delta+y = F(x, y+1) - F(x, y) \\ \Delta-y = F(x, y-1) - F(x, y) \end{cases}$$

La généralisation au cas d'une fonction quelconque du second degré est immédiate :

soit $F(x, y) = Ax^2 + By^2 + Cxy + Dx + Ey + F$

alors $\begin{cases} \Delta+x = A(2x+1) + Cy + D \\ \Delta-x = A(-2x+1) - Cy - D \\ \Delta+y = B(2y+1) + Cx + E \\ \Delta-y = B(-2y+1) - Cx - E \end{cases}$

L'évaluation de ces incréments peut se faire par calcul direct, comme indiqué dans le cas du cercle, x et y étant connus dans chaque cellule, mais également par calcul incrémental. Ainsi le calcul de $\Delta+x$ fait intervenir quatre incréments

$$\begin{cases} \delta+x = \Delta_{+(x+1)} - \Delta_{+x} = 2A \\ \delta-x = -2A \\ \delta+y = C \\ \delta-y = -C \end{cases}$$

Les incréments Δ étant calculés, le calcul de F par propagation est immédiat à partir de points remarquables.

Généralisation

Le raisonnement ci-dessus induit par récurrence la possibilité de calculer toute fonction F(x, y) dès que ses incréments Δ sont calculables. En particulier toutes les fonctions polynomiales en x et y sont calculables (Méthode de Horner).

2. Applications

Elles sont nombreuses et évidentes :

- tracé de courbes : segment, cercle, coniques etc ...
- calculs d'ombrage (bilinéaire par exemple)

III.4.1.5 - Algorithmes divers

Nous citons ici seulement pour mémoire quelques algorithmes dont la cellularisation est immédiate.

1. Algorithme de surfaces visibles

Le choix du "z-buffer" ([AT 79]) induit immédiatement une approche cellulaire : il suffit que chaque cellule ait un registre de profondeur z. l'ajout d'un nouvel objet se fera ou non suivant la relation entre le z présent et le z nouveau, et éventuellement la transparence de ce qui existe ou du nouvel objet.

2. Découpage (en deux dimensions)

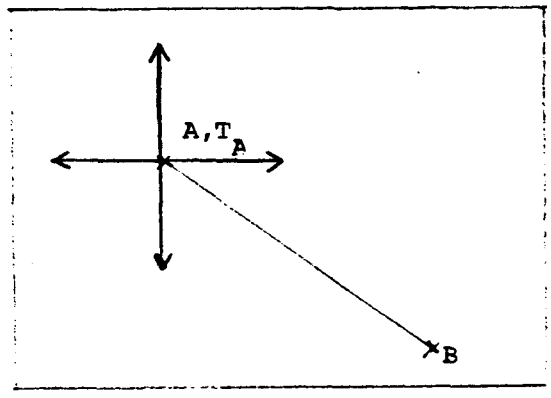
La manière la plus simple de traiter ce problème est de procéder en deux temps :

- élaboration sans découpage de l'objet à traiter
- validation des résultats dans la fenêtre de découpage

(qui pourrait être quelconque)

3. Lissage

Nous ne nous intéressons pas ici au problème du lissage par moyennage sur les voisins, qui, par son analogie avec le problème de convolution, sera traité dans le paragraphe sur le traitement d'images. Il s'agit plutôt ici de la propagation à partir d'un point A d'une valeur T_A connue, qui va influencer le voisinage de A.



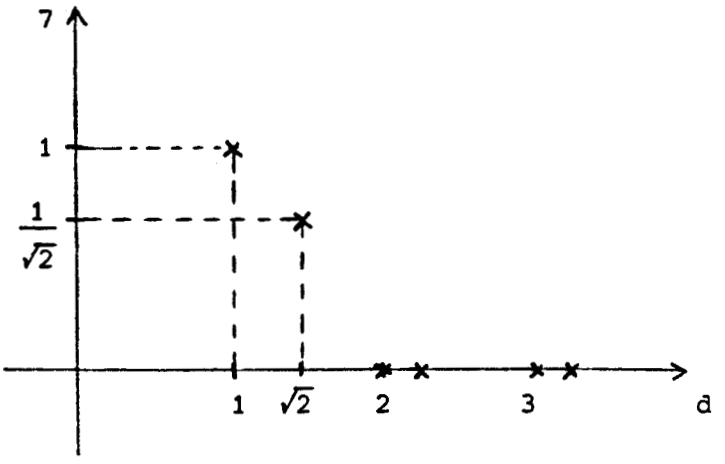
Si nous supposons que l'influence de A sur un point B est uniquement liée à la distance (réelle) entre A et B et à la valeur de T_A, il convient de propager T_A et d_x, d_y déplacements en x et y à partir de A. On en déduira facilement T_B par

$$T_B = f(T_A, \sqrt{dx^2+dy^2})$$

f étant typiquement la fonction d'atténuation avec la distance. Cette fonction peut d'ailleurs être plus complexe et en particulier non isotrope.

une fonction de lissage classique sur huit voisins s'implémente immédiatement avec une fonction f de forme simple, isotrope et donc réduite à un tableau du premier octant.

dx	dy	d	f
0	1	1	1
1	1	$\sqrt{2}$	$1/\sqrt{2}$
0	2	2	0
1	2	$\sqrt{5}$	0
2	2	$2\sqrt{2}$	0
0	3	3	0
1	3	$\sqrt{10}$	0
2	3	$\sqrt{13}$	0
3	3	$3\sqrt{2}$	0



III.4.2 - ALGORITHMES DE TRAITEMENT D'IMAGES

Les algorithmes de traitement ont pour but de produire à partir d'une image source une nouvelle image réputée contenir une information plus aisément exploitable par l'utilisateur.

III.4.2.1 - Classification

Nous retiendrons ici une classification à deux niveaux :
Le premier concerne le parallélisme, le second le voisinage concerné en termes de pixels.

Ainsi, parmi les algorithmes parallèles par nature, on peut distinguer :
- les opérations strictement locales au pixel : conversion, seuillage, correction de Gamma, opérations logiques, incrustation, opérations arithmétiques, masquages de bit
- les opérations locales, ne mettant en jeu que les voisins immédiats de chaque pixel : filtrage, lissage simples.
- les opérations globales : convolution (Laplacien, lissage, gradient), FFT, corrélations, différentiation, filtres logiques, analyse de texture, "deblurring", certains algorithmes de tri....

Certains algorithmes ne sont par ailleurs pas naturellement parallèles : ainsi en est-il par exemple des problèmes de comptage (histogramme), de moyenne, de recherche de minimum ou de maximum.

D'autres sont typiquement à propagation : étiquetage par région (équivalent au problème du remplissage), recherche de connectivité, suivi de contours, évaluation de périmètres ou surfaces ...

Nous nous intéressons d'une part aux algorithmes intrinsèquement parallèles, d'autre part à ceux qui relèvent directement d'un phénomène de propagation.

III.4.2.2 - Algorithmes parallèles

1. Les opérations pixels

Elles ne présentent pas de difficultés particulières : il n'y a pas de communications dans le réseau et chaque cellule exécute la fonction

$$y = F(x)$$

Donnons quelques exemples de telles opérations :

a) seuillage : si x > seuil | alors x
| sinon seuil (seuillage inférieur)

b) conversion / correction de gamma :

$$y = e^{\gamma x}$$

c) opérations logiques : si x_2 alors y = x
sinon y = x_2 (exemple)

2. Les opérations locales

Elles ne tiennent compte que des voisins directement connectés, i.e. de 4 voisins dans notre hypothèse.

Les applications sont donc assez limitées. Citons quelques exemples :

a) Opération logique : si $\bar{x}$ | alors
| sinon si(∃ voisin = 1) alors 1

b) Lissage restreint : $x = \frac{4x + n + e + s + w}{8}$

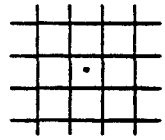
c) Laplacien : $x = -4x + n + e + x + w$

d) Translations élémentaires : x = n par exemple

3. Les opérations globales

On trouve dans cette catégorie toutes les opérations à réaliser sur chaque pixel, mettant en jeu des pixels non directement connectés. Il en est ainsi par exemple de :

a) Lissage : soit par exemple un lissage simple 3×3



$$F(x, y) = \sum_{i=-1}^{i=1} \sum_{j=-1}^{j=1} {}_{i} F(x+i, y+j)$$

ou $F(x, y) = M.F(x, y)$

b) Gradient : citons deux formules de gradient 3×3, en supposant que le voisinage du pixel i est

a	b	c
h	i	d
g	f	e

$= I$

. $|G| = \left| \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{pmatrix} I \right| + \left| \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix} I \right|$

. $Gx = (c-g) + (e-a) + 2(d-h)$

$Gy = (c-g) + (a-e) + 2(b-f)$

On obtient alors un vecteur $\vec{G}$ caractérisé par son module et son argument.

La formulation générale de ces opérations est la convolution discrète bidimensionnelle suivante :

$$G(i, j) = \sum_{k=-\infty}^{+\infty} \sum_{m=-\infty}^{+\infty} F(k, m)\ H(i-k, j-m)$$

où * $G(i, j)$ est la valeur du pixel de destination, dans une image de résolution R_i.

* $F(k, m)$ est la valeur des pixels d'origine, avec une résolution R_s.

* H est un filtre bidimensionnel séparable, de dimension R_f généralement nul pour $k \notin [k_1, k_2]$ ou $m \notin [m_1, m_2]$

d'où $G(i, j) = \sum_{k=k_1}^{k_2} \sum_{m=m_1}^{m_2} F(k, m)\ H_i(i-k) \times H_j(j-m)$

Notons quelques cas particuliers ([CRC 77])

→ 1) $R_s = 3R_i$ et $H = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$

Il s'agit d'un échantillonage avec lissage uniforme

→ $R_s = 5R_i$ et $H = \begin{bmatrix} 1/9 & 2/9 & 1/3 & 2/9 & 1/9 \\ 2/9 & 4/9 & 2/3 & 4/9 & 2/9 \\ 1/3 & 2/3 & 1 & 2/3 & 1/3 \\ 2/9 & 4/9 & 2/3 & 4/9 & 2/9 \\ 1/9 & 2/9 & 1/3 & 2/9 & 1/9 \end{bmatrix}$

Il s'agit d'un échantillonage avec filtrage triangulaire.

→ 3) Si $R_s = \infty$ (F continue)

* H_i = _/_
  ←→ 1 pixel

  (image naturelle et filtrage simple)

* H_i = _/__
  ←→ 2 pixels

  (image naturelle et filtrage avec recouvrement)

Le nombre d'opérations à effectuer par pixel est égal à

$$n = R_f^2 \times R_s^2$$

Dans le cas particulier, mais très fréquent de la Transformée de Fourier Rapide, on a

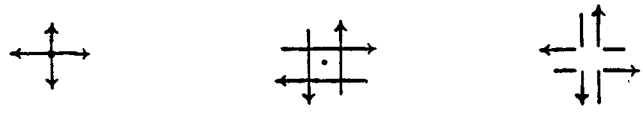
$$R_i = R_s = R_f$$

Le filtre étant particulier, on peut théoriquement atteindre un nombre d'opération n égal à $2R_s^2 \log_2 R_s$

ex : $R_s = 1024$ => $n \simeq 20$ M opérations

En réalité, ce qui nous intéresse dans ce problème n'est pas tant la formule de calcul, mais plutôt la nécessité pour le processeur responsable de l'évaluation d'un pixel donné de connaître un (grand) nombre de pixels voisins. Nous choisissons d'effectuer tous les calculs relatifs à un pixel à l'intérieur de ce processeur ; il suffit donc de transmettre à ce pixel les valeurs des voisins nécessaires. Nous avons proposé pour cela un mode de diffusion hélicoïdal qui a les caractéristiques que nous rappelons ci-après :

1) il y a trois diffusions élémentaires différentes



(0) (1) (2)

(0) est la diffusion initiale d'une valeur d'une cellule à ses quatre voisines

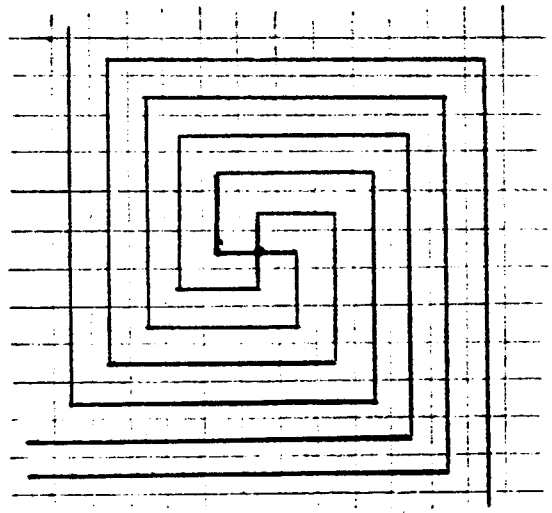
(1) est la diffusion en ligne droite

(2) est la diffusion à angle droit.

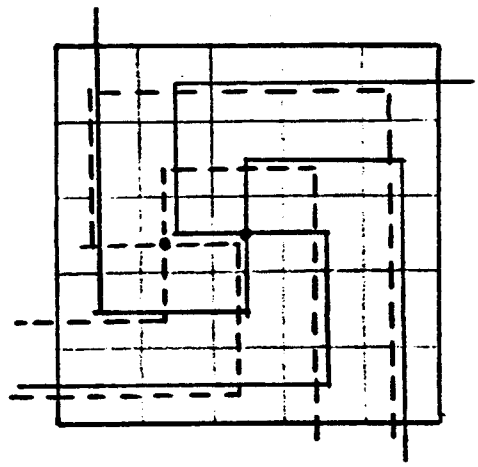
2) Pour diffuser la valeur de chaque cellule à l'ensemble des cellules voisines d'un carré (2n+1)×(2n+1) centré en cette cellule, il faut effectuer l'algorithme suivant sur chaque cellule

```
DIFFUSION :
   pour i = 1 à n
      faire
           (1) ; /* (0) au premier passage */
           (2) ;
           [2i-2] * (1) ;
      fait ;
```

Le schéma de propagation pour une cellule est le suivant



Le schéma simplifié suivant indique les parcours pour des cellules différentes



En réalité, on constate aisément qu'il n'y a pas de conflits de communications entre cellules.

3) Performances

La communication complète de valeurs sur un domaine $(2n+1)\times(2n+1)$ se fait en

$$T = n(n+1) \text{ cycles}$$

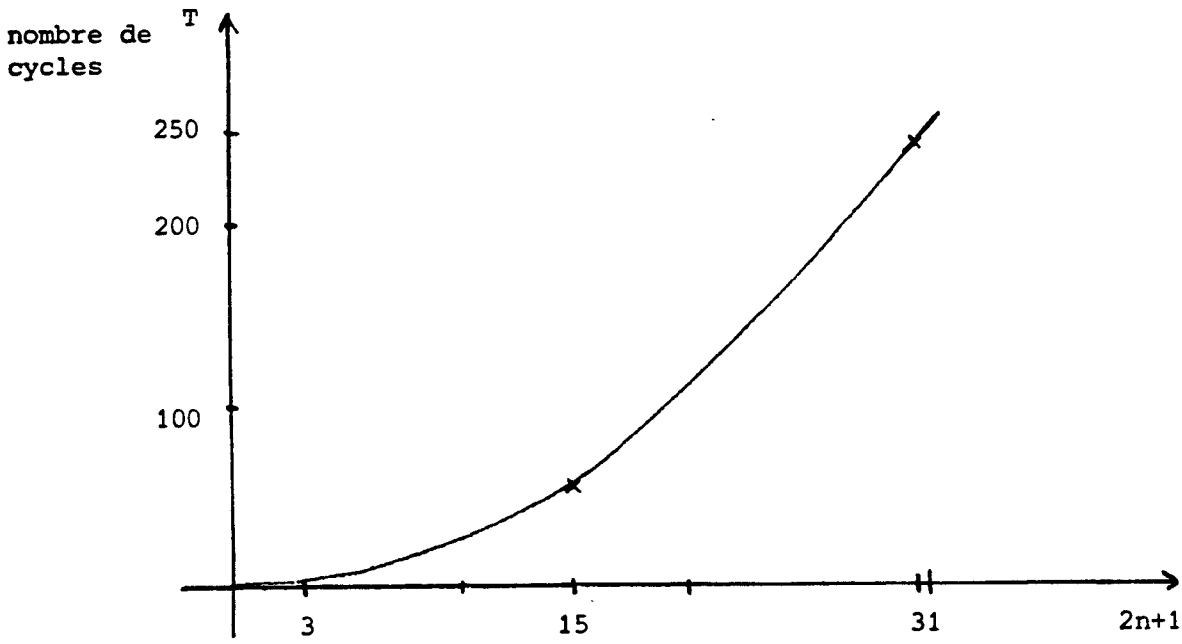
* Démonstration

1) $T = \sum_{1}^{n} 2i = n(n+1)$ d'après l'algorithme

ou

2) $T = \frac{(2n+1)^2-1}{4}$ d'après la propagation

Le graphique suivant résume les performances $T = f(2n+1)$



Notons les deux valeurs particulières

$T = 2$ pour une fenêtre 3×3

$T \simeq 256K$ pour une fenêtre 1023×1023

Une distribution arborescente quaternaire (toute théorique, car bien difficile à câbler !) donnerait

$T \simeq n \log_4 n$

soit $T = 2$ pour une fenêtre 3×3

et $T \simeq 4K$ pour une fenêtre 1K×1K

4) Chaque cellule (x, y) reçoit successivement à chaque cycle quatre valeurs, suivant l'ordre :

cycle	N	E	S	W
1	x, y+1	x+1, y	x, y-1	x-1, y
2	x-1, y+1	x+1, y+1	x+1, y-1	x-1, y-1
3	x-1, y+2	x+2, y+1	x-1, y-2	x-2, y-1
4	x-2, y	x, y+2	x+2, y	x, y-2
5	etc..	...	...	...
...				

4. Opérations à propagation (à partir d'un ou plusieurs centres)

Il s'agit d'algorithmes à propagation non prévisible : le choix de la direction dépend du contenu de l'image. Nous citerons deux exemples caractéristiques : le suivi de contour et l'étiquetage de région.

a) Suivi de contour :

Les utilisations en sont évidentes en traitement d'image, c'est bien souvent une étape essentielle d'un processus de reconnaissance de forme. Le résultat peut être un marquage des cellules ou l'extraction du code (de Freeman) du contour dans divers buts (reconnaissance, détermination de paramètres géométriques, etc ...) L'algorithme consiste à partir d'une cellule de contour et à chercher parmi ses voisines une cellule de contour, etc .. de proche en proche. Si plusieurs cellules sont candidates, on peut

1) essayer toutes les pistes, et l'on aboutit à un arbre de possibilités, à résoudre par des méthodes d'intelligence artificielle
2) en choisir une seule.

Dans chaque cas, il convient, de manière à éviter les cycles, de marquer sa route.

b) Etiquetage de région

Il s'agit en partant d'un point à l'intérieur d'une région (produite par suivi de contour), de marquer toute la région. On voit ici l'identité de ce problème avec celui du remplissage en synthèse d'images, auquel on se réfèrera.

5. Opération à propagation totale

Un exemple typique est celui de la recherche du maximum d'une image : il faut que toutes les valeurs soient comparées pour déterminer la valeur du maximum. Nous proposons un procédé parallèle et un procédé à décalage, qui prennent le même temps si l'opération est la seule à s'effectuer.

1) Décalage (en colonne)

* On décale progressivement chaque colonne vers la gauche en partant de la droite, en effectuant l'algorithme

$$\underline{si}\ x > e \begin{cases} \underline{alors}\ w = x \\ \underline{sinon}\ w = e \end{cases}$$

* On obtient donc dans la dernière colonne de gauche les maxima des lignes. Il suffit de réitérer le processus en partant du bas de cette colonne.

$$\underline{si}\ x > s \begin{cases} \underline{alors}\ n = x \\ \underline{sinon}\ n = s \end{cases}$$

et l'on obtient dans la cellule en haut à gauche le maximum des valeurs du tableau.

Le temps de calcul est $2(n-1)\theta$ si on a un réseau $n \times n$ de temps de cycle θ ; le nombre de comparaisons effectuées est de n^2-1. Notons qu'à un moment donné une seule colonne est active et que le nombre total de comparaisons effectuées est égal au nombre prédit par la théorie (recherche du maximum parmi $N = n^2$ valeurs).

2) Propagation totale

Chaque cellule exécute $2(n-1)$ l'algorithme

$$x = \sup(x, n, e, s, w)$$

Il est clair qu'au bout d'un temps fini, égal à $2(n-1)\theta$, toutes les cellules auront la valeur du maximum. Mais à un moment donné, toutes les cellules sont actives (fonctionnement SIMD vrai). De plus le nombre total de comparaisons effectuées est d'environ $8(n-1)\ n^2$, ce qui est très supérieur au nombre fourni par la théorie. Cependant la valeur du maximum est alors connue de chaque cellule et non pas d'une seule comme dans le cas précédent. D'autres algorithmes plus ou moins efficaces ou complexes peuvent être trouvés pour ce problème ou pour des problèmes analogues (moyenne, histogramme..)

III.4.3 - ALGORITHMES DE MANIPULATIONS D'IMAGES

Il s'agit d'algorithmes ne modifiant que la structure géométrique de l'image, sans tenir compte du contenu. Pour l'essentiel sont concernés les transformations géométriques affines.

III.4.3.1 - Translation

Elle se traduit sur l'écran par ce que l'on appelle généralement "panoramique" (translation horizontale) ou "défilement" (scrolling, ou translation verticale).

En supposant que le réseau est torique, i.e. fermé dans les deux directions, une translation (Δx, Δy) peut se réaliser par des translations unitaires successives en x et en y, synchrones (i.e. en SIMD vrai).

1. Commande

On initialise les registres de tout le réseau de façon suivante :

R(2) = Δx (supposé > 0)

R(3) = Δy (supposé > 0)

R(1) est supposé contenir la valeur à transférer. Toutes les cellules sont activées simultanément.

2. Exécution

```
TRANS : EST.W = R(1)
    |   pour I = 1 à R(2) /* translation en x */
    |       |  faire EST.W = W ;
    |       |  fait
    |   NORD.S = W
    |   pour I = 1 à R(3) /* translation en y */
    |       |  faire NORD.S = S ;
    |       |  fait
    |   R(1) = S ; /* récupération de la valeur */
    |   ETAT = 0 ;
```

Notons une autre solution, qui d'ailleurs peut s'appliquer à d'autres cas, consistant à renuméroter le réseau, soit effectivement, soit seulement à la sortie vers l'affichage.

4.3.2 - Rotation

Un certain nombre de rotations se traduisent très simplement en renumérotant les cellules : 90°, 180°, 270°. Par contre leur traitement à l'aide de la propagation dans le réseau est difficile. Encore plus difficile est le traitement du cas général, car chaque cellule (ou presque) a un mouvement différent des voisines.

Pour une rotation θ de centre C (x_o, y_o) une cellule (x, y) devra effectuer une translation

$$T \begin{cases} Tx = (x-x_o)\ (\cos\theta-1) - (y-y_o)\ \sin\theta \\ \\ Ty = (x-x_o)\ \sin\theta - (y-y_o)\ (1-\cos\theta) \end{cases}$$

Deux cellules voisines effectueront donc des translations différant de

$$\begin{cases} \Delta_x\ T_x = \cos\theta-1 \\ \Delta_y\ T_y = -\sin\theta \\ \Delta_x\ T_y = \sin\theta \\ \Delta_y\ T_y = \cos\theta-1 \end{cases}$$

Ceci permet d'évaluer le vecteur de translation de chaque cellule. Bien entendu cette méthode approximative n'est ni symétrique ($T(\theta).T(-\theta) \neq I$), ni transitive ($T(\alpha).T(\beta) \neq T(\alpha+\beta)$).

III.4.5 - SYMETRIES

Une fois encore, les symétries remarquables se traduisent par une renumérotation des cellules : ce sont

- les symétries en x ou y (axe vertical ou horizontal)
- les symétries par rapport à un centre
- les symétries par rapport aux diagonales

Les autres n'ont pas de solution immédiate.

III.4.6 - MISE A L'ECHELLE (HOMOTHETIE)

Cette transformation, qui se traduit par un "zoom", présente deux cas suivant que le facteur de zoom est ou non supérieur à 1.

1. $k > 1$: agrandissement

Si une solution par renumérotation existe (analogue à la modification d'horloge des zooms câblés), il existe aussi une possibilité de mise à l'échelle effective.

Si $C(x_c, y_c)$ est le centre de zoom, il convient de translater toute cellule (x_i, y_i) de

$$T_i = \begin{cases} (x_i - x_c) \times k \\ \\ (y_i - y_c) \times k \end{cases}$$

puis éventuellement de lisser le résultat obtenu.

2. $k < 1$: rapetissement

On a de même une solution brutale consistant à prendre 1 cellule tous les 1/k. On choisira plutôt de moyenner de 1/k en 1/k à partir des 1/k × 1/k pixels voisins, puis de comprimer l'image par translation

$$T_i = \begin{cases} (x_i - x_c) \times k \\ \\ (y_i - y_c) \times k \end{cases}$$

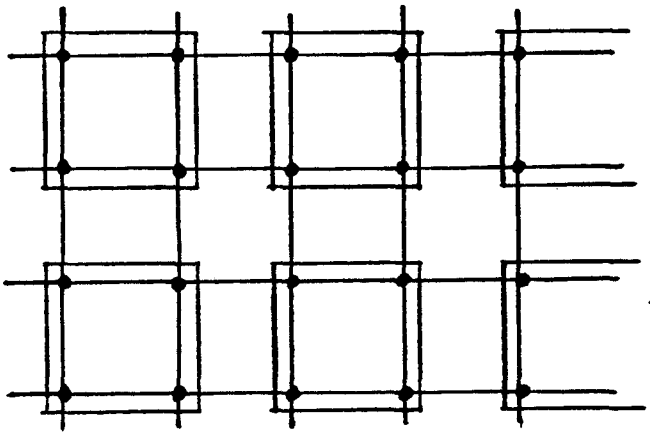
III.4.7 - REMARQUES

Les divers algorithmes rencontrés, et en particulier les algorithmes de rotation, conduisent à se poser un problème de format et de méthode de calcul. Une réalisation triviale d'opérateurs sur entier se heurte à des difficultés importantes, et il convient de prévoir une implémentation plus intelligente de certains de ces opérateurs, telle que le passage aux logarithmes ou l'utilisation de PROM ou PLA.

On notera de plus les problèmes sous-jacents à la rotation et à l'homothétie (précision des calculs, algorithme non uniforme, parallèlisme très faible, effets de bords ...).

III.5 - UNE MACHINE REELLE A 2 NIVEAUX

Il est peu réaliste d'envisager, comme nous l'avons fait, une architecture à un seul niveau, i.e. un tableau homogène de n×n processeurs tous identiques. En effet, la technologie, et la faisabilité nous imposent de réunir plusieurs de ces cellules dans un même boitier.



Il nous faut donc distinguer deux niveaux :

- le niveau microscopique, celui des cellules logiques, correspondant à l'étude déjà faite
- le niveau macroscopique, celui des composants.

III.5.1 - LE NIVEAU MICROSCOPIQUE

Il n'y a pas de différence essentielle avec l'étude déjà menée : chaque cellule dispose de :

- une horloge commune
- une Unité Arithmétique et logique
- des registres de travail
- des registres de communications
- d'une unité assurant le contrôle (µ.programme par exemple).

A l'intérieur d'un même boîtier,les problèmes de connexions peuvent être négligés, sauf en ce qui concerne la distribution des signaux communs.

III.5.2 - LE NIVEAU MACROSCOPIQUE

La machine est vue comme un ensemble de composants identiques reliés entre eux. Chaque composant assure l'interface entre l'extérieur, i.e. les autres composants et le système de contrôle extérieur, et l'intérieur, i.e. le mini-réseau de cellules.

La question fondamentale qui se pose alors est la suivante : y-a-t-il deux niveaux d'algorithmes, un au niveau composant, un au niveau pixel ?

A cette vaste question, nous donnerons seulement les indications suivantes :

* s'il existe un niveau composant, il ne peut être que partiel et concerner des aspects précis ; c'est non une innovation, mais une facilité
* au niveau contrôle, le niveau composant permet des simplifications notables (1 seule UCC, une seule mémoire de commandes, un seul détecteur de fin d'activité, etc ...).
* les différences semblent surtout porter sur l'implémentation des algorithmes et non sur leurs principes mêmes : certaines solutions sont viables dans un composant, non viables pour un système (horloges, synchronisation, voies de communications ...).

III.5.3 - INTEGRATION

La question qui se pose est de déterminer combien de cellules pourraient être intégrées dans un seul composant. Pour ce faire, il convient à la fois d'examiner à la fois la complexité d'une cellule de base et l'évolution de la technologie.

III.5.3.1 - La technologie

Déjà évoquée dans le chapitre I.2, nous savons que deux approches sont possibles :

a) le pré-diffusé (ou "semi-custom"), qui propose d'ores et déjà des composants de 1000 à 5000 portes, voire 10000, avec des retards par portes allant de 15 à 0,5 ns. Les fréquences d'horloge possibles vont de 5 MHz (NMOS) à 2000 MHz (ECL)

b) le "sur-mesure" (ou "full-custom"), beaucoup plus coûteux, mais qui réalise déjà des composants de plus de 400000 transistors.

Excluant cette deuxième hypothèse, nous supposerons que nous pouvons réaliser un circuit prédiffusé de 20000 portes, fonctionnant à 100 MHz, disposant de 256 pattes d'accès. Un tel circuit paraît réalisable entre 1985 et 1990, probablement en technologie CMOS-5.

III.5.3.2 - Complexité de la cellule de base

Une telle évaluation est très difficile à faire. Si nous comparons cette cellule à un µ-processeur classique (8080), nous trouvons une complexité plus faible (ALU et registres moins sophistiqués, adressage plus simpl-

Si nous comparons maintenant à une ALU classique (74181), il faut ajouter la complexité des registres et de la µprogrammation.

Nous avancerons donc, sous réserves, le chiffre de 2000 portes par cellules, auquel on ajoutera quelques centaines de mots de µprogrammes. Le prédiffusé nous permettrait donc d'intégrer quatre cellules par boîtier, ce qui est peu ; le "sur-mesure" nous conduirait à 80 cellules par boîtier (disons 64 !), ce qui est plus raisonnable.

Si l'on choisit une transmission série entre composants, il faut, dans ce dernier cas, 64 pattes de communications ; ce qui est très raisonnable. Quant au temps de communication, pour des mots de 8 bits, il serait de 100 ns, soit un débit entre boîtiers de 10 M mots/sec. Pour un réseau de 512×512 pixels, soit 64×64 boîtiers, le temps de propagation d'un bout à l'autre en diagonale serait donc de 128×100 ns, soit 12,8 µs.

III.5.3.3 - Solutions de compromis

Elles sont de deux ordres : le premier, technologique, est de faire appel à des composants existants ; ainsi le transputer (INMOS) semble-t-il tout à fait convenir à ce type de machine. Le second, logique, serait de ne câbler qu'une partie du réseau, affecté statistiquement où dynamiquement à une partie de l'image. Ce problème difficile d'affectation et de reconfiguration ne sera pas traité ici.

CONCLUSIONS

Cette étude montre qu'il est possible d'envisager la construction de machine cellulaires simples pour la synthèse ou le traitement des images. Certaines des hypothèses faites pourraient être remises en cause : ainsi notre approche macro-SIMD associée à une propagation uniforme ou non de données dans le réseau pourrait être remplacée par une approche MIMD, à propagation du contrôle dans le réseau. Mais dans ce cas, il apparaît difficile de traiter les problèmes fondamentalement SIMD (filtrage ...).

De même l'espace cellulaire à 4 voisins pourrait se voir substituer un espace à 8 voisins, voire 6 ou 3, comme cela devient la mode en traitement d'image.

Notons également que des composants cellulaires, tels le Transputer existent maintenant, et que de toutes façons les VLSI favorisent ce genre d'architecture ; enfin pourquoi ne pas imaginer des réseaux cellulaires 3D pour les décennies à venir ...

L'étude faite et la simulation réalisée nous permet toutefois de montrer deux points essentiels :

. cette classe d'architecture ne peut se satisfaire des algorithmes classiques

. elle met en oeuvre des composants tous identiques, ce qui est un argument majeur pour le développement de machines basées sur ces concepts.

ANNNEXE 1 : LOGIQUE CELLULAIRE DE REMPLISSAGE

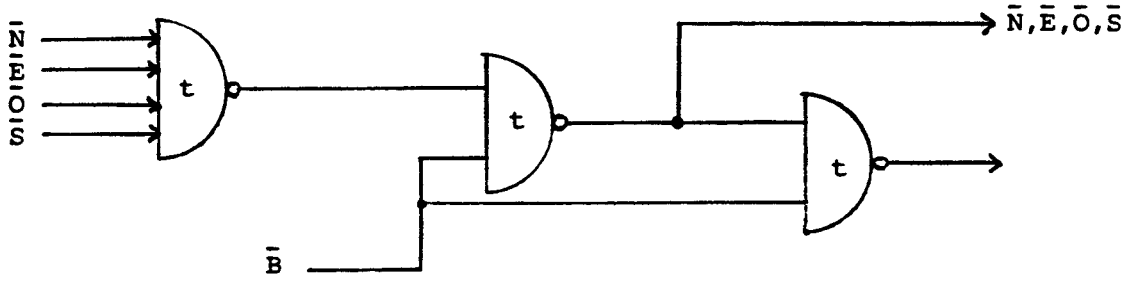
1. PRINCIPE

De l'algorithme décrit en §III.2.2.3, nous déduisons aisément une logique cellulaire, qui est bien entendu spécifique à ce problème de remplissage. Soit B un registre de "bord", initialisé au cours d'une phase de fonctionnement précédente (par exemple tracé de contour), et S est une sortie "intérieur" indiquant si l'on se trouve à l'intérieur du contour ou sur celui-ci ; dans ce cas nous avons

$$I = (N + E + S + O) \cdot \bar{B}$$

$$S = I + B$$

Nous obtenons donc le schéma logique suivant pour une cellule :



Le temps d'élaboration d'un pixel est de 3t, mais le temps intervenant dans la propagation est de 2t, si l'on suppose que t est le retard amené par une porte NAND.

Si ℓ est la "longueur de propagation", i.e. le chemin le plus long pour atteindre tous les points de l'intérieur cherché à partir d'un point source, la durée du remplissage est

$$\tau = \ell \times 2t.$$

En prenant t = 3ns (technologie très classique) et ℓ = 512 (ce qui correspond à remplir un pavé 512×512 à partir du centre), on obtient $\tau \# 3\mu s$.

Notons au passage qu'un tel remplissage supposé idéal, sur une mémoire à accès aléatoire ayant un temps d'accès de 100 ns, durerait 100 ns×512×512 = 25ms.

2. REALISATION

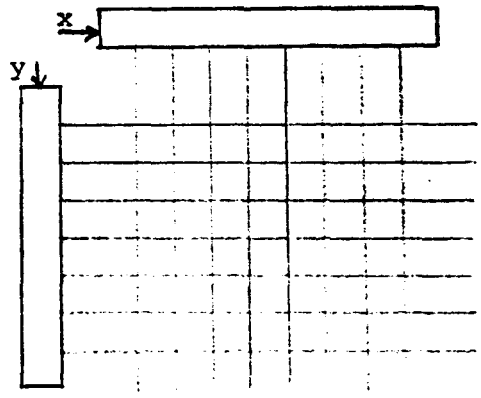
Si la réalisation du "coeur" logique décrit ci-dessus ne pose pas de problème, il convient de noter d'ores et déjà les problèmes liés aux entrées-sorties :

. initialisation du registre B

. récupération de S

. démarrage du processus de remplissage par **désignation** d'un (ou plusieurs) point(s) intérieur(s).

En imaginant de pouvoir sélectionner une cellule par décodage x et y suivant le schéma :



ainsi que e pouvoir lire ou écrire une valeur booléenne, alors on peut estimer le nombre de connexions par cellule à :

. 2×4 pour les communications entre voisines

. 1+1 pour la sélection de la cellule

. 1 pour écriture/lecture

. 1 pour la valeur booléenne.

Soit un total de 12 connexion/cellule.

Si l'on groupe plusieurs cellules par boîtier, avec bien sûr un décodage complémentaire au niveau du boîtier, on obtient les résultats suivants :

taille du boitier	1×1	2×2	4×4	8×8	16×16	n×n, ($n=2^p$)
communications inter cellules	8	16	32	64	128	8n
sélection cellule	2	4	6	8	10	2(p+1)
L/E + valeur	2	2	2	2	2	2
total	12	22	40	74	140	8n + 2p + 4

Compte tenu des possibilités de la technologie (en nombre de portes et de connexions par boîtier), une solution 8×8 est tout à fait envisageable. Un réseau logique cellulaire de remplissage 512×512 comporterait donc 64×64 = 4096 composants, tous identiques, câblés de manière entièrement répétitive.

Ces problèmes technologiques seront évoqués et complétés ultérieurement.

3. SIMULATION

Une simulation, que l'on trouvera ci-après, a été réalisée, sur la base d'un algorithme de "jeu de la vie" modifié.

ANNEXE 2 : SIMULATION

1. PRINCIPE

Il s'agit de simuler les fonctionnalités du réseau, et non de simuler finement son comportement (au niveau logique). Nous cherchons à la fois à valider le choix du type de fonctionnement (SIMD et propagation de données) et les algorithmes élémentaires de synthèse proposés. La machine étant synchrone, nous avons choisi d'avoir deux versions de l'état du réseau, une ancienne et une nouvelle ; l'exécution d'une opération sur le réseau produisant séquentiellement à partir de celui-ci un nouveau réseau. On assure ainsi la cohérence des données pour une opération particulière.

Chaque cellule a la structure suivante :

- 4 registres de communications N, E, S, W
- un registre d'état de communication I N
- 6 registres de travail R(i)
- 2 registres de coordonnées X et Y
- 1 registre d'ETAT.

Le simulateur permet de lire et d'écrire dans chaque registre de chaque cellule, individuellement ou par pavé. Il permet de simuler totalement la partie contrôle du réseau.

Ont été implémentés les algorithmes suivants

1) tracé de segment
2) tracé de cercle
3) remplissage

2. PROGRAMMES ET RESULTATS

On trouvera ci-après les programmes de simulation et quelques résultats obtenus.

PL/M-86 COMPILER ARRAY 15/11/83

```
SERIES-III PL/M-86 V2.0 COMPILATION OF MODULE ARRAY
OBJECT MODULE PLACED IN :F4:CELL.OBJ
COMPILER INVOKED BY:  :F2:PLM86.86 :F3:CELL.PLM PRINT(:F5:CELL.LST) OBJECT(:F4:CELL.OBJ) DEBUG LARGE SYMBOLS XREF


   1          ARRAY:DO;

   2    1     DECLARE T LITERALLY '16';   /* taille du tableau */
   3    1     DECLARE CELL LITERALLY 'STRUCTURE (N BYTE,E BYTE,S BYTE,W BYTE,
                                                 R(6) BYTE,
                                                 X BYTE,Y BYTE,
                                                 ETAT BYTE)';
   4    1     DECLARE OLDTAB(400) CELL;
   5    1     DECLARE NEWTAB(400) CELL;

   6    1     DECLARE OLDPTR POINTER;
   7    1     DECLARE NEWPTR POINTER;
   8    1     DECLARE (NPTR,EPTR,SPTR,WPTR) POINTER;

   9    1     DECLARE PIXEL BASED OLDPTR CELL;
  10    1     DECLARE NPIXEL BASED NEWPTR CELL;
  11    1     DECLARE NORD BASED NPTR CELL,
                      EST BASED EPTR CELL,
                      SUD BASED SPTR CELL,
                      WEST BASED WPTR CELL;


  12    1     DECLARE (A,COMPTE) ADDRESS;
  13    1     DECLARE (I,J,K,REG,VAL,IN,ST) BYTE;

  14    1     CI: PROCEDURE BYTE EXTERNAL;
  15    2         END CI;
  16    1     CO: PROCEDURE(CHAR) EXTERNAL;
  17    2         DECLARE CHAR BYTE;
  18    2         END CO;
  19    1     PRINT:PROCEDURE(OCT);
  20    2         DECLARE (OCT,DIZ,CENT) BYTE;
  21    2         IF OCT)127 THEN DO;
  23    3                         CALL CO('-');
  24    3                         OCT=-OCT;
  25    3                         END;
  26    2                   ELSE CALL CO(20H);
  27    2         IF (CENT:=OCT/100)=0 THEN CALL CO(20H);
  29    2                              ELSE CALL CO(CENT OR 30H);
  30    2         IF ((DIZ:=(OCT-100*CENT)/10)()0 OR (CENT()0)) THEN CALL CO (DIZ OR 30H);
  32    2                                                       ELSE CALL CO(20H);
  33    2         CALL CO((OCT-10*DIZ-100*CENT) OR 30H);
  34    2         CALL CO(20H);
  35    2         END PRINT;

            $EJECT
```

```
 36   1      INREG:PROCEDURE BYTE;
 37   2          DECLARE R BYTE;
 38   2          R=CI AND 07FH;CALL CO(R);CALL CO(0DH);CALL CO(0AH);

 42   2          IF R='N' THEN RETURN 6;
 44   2          IF R='E' THEN RETURN 7;
 46   2          IF R='S' THEN RETURN 8;
 48   2          IF R='W' THEN RETURN 9;
 50   2          IF R='X' THEN RETURN 10;
 52   2          IF R='Y' THEN RETURN 11;
 54   2          IF R='A' THEN RETURN 12;
 56   2          IF (R:=R-30H)<6 THEN RETURN R;
 58   2          END INREG;

             $EJECT
```

```
  68   1      INIT:PROCEDURE;          /*  initialisation */
  69   2          NPIXEL.N,NPIXEL.E,NPIXEL.S,NPIXEL.W=0;
  70   2          NPIXEL.X=J;
  71   2          NPIXEL.Y=I;
  72   2          NPIXEL.ETAT=0;
  73   2          DO K=0 TO 5;
  74   3              NPIXEL.R(K)=0;
  75   3              END;
  76   2          END INIT;

             $EJECT
```

15/11/83

PL/M-86 COMPILER ARRAY

```
 77   1       RAZ:PROCEDURE(K,V);
 78   2           DECLARE (K,V) BYTE;
 79   2           IF K<6 THEN NPIXEL.R(K)=V;
 81   2               ELSE DO CASE (K-6);
 82   3                        NPIXEL.N=V;
 83   3                        NPIXEL.E=V;
 84   3                        NPIXEL.S=V;
 85   3                        NPIXEL.W=V;
 86   3                        NPIXEL.X=J;
 87   3                        NPIXEL.Y=I;
 88   3                        NPIXEL.ETAT=V;
 89   3                        END;
 90   2           END RAZ;

              $EJECT
```

```
 91   1      RECOPIE:PROCEDURE;          /* recopie new dans old */
 92   2          COMPTE=0;
 93   2          DO I=0 TO T-1;
 94   3              DO J=0 TO T-1;
 95   4                  OLDPTR=@OLDTAB(I*T+J);
 96   4                  NEWPTR=@NEWTAB(I*T+J);
 97   4                  PIXEL.N=NPIXEL.N;
 98   4                  PIXEL.E=NPIXEL.E;
 99   4                  PIXEL.S=NPIXEL.S;
100   4                  PIXEL.W=NPIXEL.W;
101   4                  DO K=0 TO 5;
102   5                      PIXEL.R(K)=NPIXEL.R(K);
103   5                      END;
104   4                  PIXEL.ETAT=NPIXEL.ETAT;
105   4                  IF PIXEL.ETAT =2 THEN COMPTE=COMPTE+1;
107   4                  END;
108   3              END;
109   2          END RECOPIE;

          $EJECT
```

15/11/83

PL/M-86 COMPILER ARRAY

```
 110   1      AFF:PROCEDURE(K);
 111   2          DECLARE K BYTE; /* reg a afficher */
 112   2          IF K<6 THEN CALL PRINT(PIXEL.R(K));
 114   2              ELSE DO CASE (K-6);
 115   3                   CALL PRINT(PIXEL.N);
 116   3                   CALL PRINT(PIXEL.E);
 117   3                   CALL PRINT(PIXEL.S);
 118   3                   CALL PRINT(PIXEL.W);
 119   3                   CALL PRINT(NPIXEL.X);
 120   3                   CALL PRINT(NPIXEL.Y);
 121   3                   CALL PRINT(PIXEL.ETAT);
 122   3                   END;
 123   2          IF (J=T-1) THEN CALL CO(0DH);
 125   2          END AFF;

             $EJECT
```

```
126   1       CONTOUR:PROCEDURE;  /*inscription d'une suite de pixels dans r(0) */
127   2           DO WHILE OFFH;
128   3               I=INVAL;
129   3               IF I=OFH THEN GO TO SUITE;
131   3               J=INVAL;
132   3               NEWTAB(I*T+J).R(0)=1;
133   3               END;
134   2           SUITE:;
135   2           END CONTOUR;

              $EJECT
```

15/11/83

PL/M-86 COMPILER ARRAY

```
136   1     REMPLIR:PROCEDURE;
137   2         IN=PIXEL.N OR PIXEL.S OR PIXEL.E OR PIXEL.W;
138   2         DO CASE PIXEL.ETAT;
139   3             NPIXEL.ETAT,NORD.S,EST.W,SUD.N,WEST.E=0;
140   3             DO CASE IN;
141   4                 DO;
142   5                     NORD.S,EST.W,SUD.N,WEST.E=0;
143   5                     NPIXEL.R(1)=PIXEL.R(0);
144   5                     NPIXEL.ETAT=1;
145   5                     END;
146   4                 DO;
147   5                     NORD.S,EST.W,SUD.N,WEST.E=1-PIXEL.R(0);
148   5                     NPIXEL.R(1)=1;
149   5                     NPIXEL.ETAT=0;
150   5                     END;
151   4                 END;
152   3             DO;
153   4                 NPIXEL.R(1)=1;
154   4                 NPIXEL.ETAT=0;
155   4                 NORD.S,EST.W,SUD.N,WEST.E=1-PIXEL.R(0);
156   4                 END;
157   3             END;
158   2         END REMPLIR;

          $EJECT
```

15/11/83

PL/M-86 COMPILER ARRAY

```
 162   1       SEGMENT:PROCEDURE;  /* procedure qui trace un segment */
                   /*  R(0)=1 sur le segment
                       R(1) N U
                       R(2)=2*DX
                       R(3)=2*DY
                       R(4)=Erreur
                       R(5)=uDXu ou uDYu */
 163   2           IN=(ST:=PIXEL.N OR PIXEL.E OR PIXEL.S OR PIXEL.W) AND OFEH;
 164   2           ST=ST AND 01H;
 165   2           DO CASE PIXEL.ETAT;
 166   3               DO;
 167   4                   NPIXEL.ETAT=0;
 168   4                   NORD.S,EST.W,SUD.N,WEST.E=0;
 169   4                   NPIXEL.R(0)=PIXEL.R(0);
 170   4                   END;
 171   3               DO;
 172   4                   IF ST=0 THEN DO;
 174   5                                NPIXEL.ETAT=1;
 175   5                                NORD.S,EST.W,SUD.N,WEST.E=0;
 176   5                                NPIXEL.R(0)=PIXEL.R(0);
 177   5                                END;
 178   4                           ELSE DO;
 179   5                                NPIXEL.ETAT=0;
 180   5                                NPIXEL.R(4)=IN;
 181   5                                EST.W=(IN+PIXEL.R(3)) OR 01H;
 182   5                                WEST.E=(IN-PIXEL.R(3)) OR 01H;
 183   5                                NORD.S=(IN+PIXEL.R(2)) OR 01H;
 184   5                                SUD.N=(IN-PIXEL.R(2)) OR 01H;
 185   5                                IF IN>127 THEN IN=-IN;
 187   5                                IF IN<=PIXEL.R(5) THEN NPIXEL.R(0)=1;
 189   5                                                  ELSE NPIXEL.R(0)=PIXEL.R(0);
 190   5                                IF IN>90 THEN NORD.S,EST.W,SUD.N,WEST.E=0;/*test de debordement*/
 192   5                                END;
 193   4                   END;
 194   3               DO;
 195   4                   NPIXEL.ETAT=0;
 196   4                   NPIXEL.R(4)=PIXEL.R(4);/* erreur initiale */
 197   4                   EST.W=PIXEL.R(3) OR 01H;
 198   4                   WEST.E=-PIXEL.R(3) OR 01H;
 199   4                   NORD.S=PIXEL.R(2) OR 01H;
 200   4                   SUD.N=-PIXEL.R(2) OR 01H;
 201   4                   NPIXEL.R(0)=1;
 202   4                   END;
 203   3               END;
 204   2           END SEGMENT;

               $EJECT
```

```
205   1     CERCLE:PROCEDURE;   /* procedure qui trace un cercle */
              /*  R(0)=1 sur le cercle
                  R(1) N U
                  R(2)=4*DX
                  R(3)=4*DY
                  R(4)=Erreur=-R*R au centre;0 sur le cercle
                  R(5)=test =4*R */
206   2           IN=(ST:=PIXEL.N OR PIXEL.E OR PIXEL.S OR PIXEL.W) AND OFEH;
207   2           ST=ST AND 01H;
208   2           DO CASE PIXEL.ETAT;
209   3               DO;
210   4                   NPIXEL.ETAT=0;
211   4                   NORD.S,EST.W,SUD.N,WEST.E=0;
212   4                   NPIXEL.R(0)=PIXEL.R(0);
213   4                   END;
214   3               DO;
215   4                   IF ST=0 THEN DO;
217   5                                   NPIXEL.ETAT=1;
218   5                                   NORD.S,EST.W,SUD.N,WEST.E=0;
219   5                                   NPIXEL.R(0)=PIXEL.R(0);
220   5                                   END;
221   4                           ELSE DO;
222   5                                   NPIXEL.ETAT=0;
223   5                                   NPIXEL.R(4)=IN;
224   5                                   EST.W=(IN+PIXEL.R(3)+2) OR 01H;
225   5                                   WEST.E=(IN-PIXEL.R(3)+2) OR 01H;
226   5                                   NORD.S=(IN+PIXEL.R(2)+2) OR 01H;
227   5                                   SUD.N=(IN-PIXEL.R(2)+2) OR 01H;
228   5                                   IF IN>127 THEN IN=-IN;
229   5                                   IF IN<=PIXEL.R(5) THEN NPIXEL.R(0)=1;
230   5                                                     ELSE NPIXEL.R(0)=PIXEL.R(0);
233   5                                   IF IN>90 THEN NORD.S,EST.W,SUD.N,WEST.E=0;/*test de debordement*/
235   5                                   END;
236   4                   END;
237   3               DO;
238   4                   NPIXEL.ETAT=0;
239   4                   NPIXEL.R(4)=PIXEL.R(4);/* erreur initiale */
240   4                   EST.W=(PIXEL.R(3)+2) OR 01H;
241   4                   WEST.E=(-PIXEL.R(3)+2) OR 01H;
242   4                   NORD.S=(PIXEL.R(2)+2) OR 01H;
243   4                   SUD.N=(-PIXEL.R(2)+2) OR 01H;
244   4                   NPIXEL.R(0)=1;
245   4                   END;
246   3               END;
247   2           END CERCLE;

            $EJECT
```

15/11/83

PL/M-86 COMPILER ARRAY

```
248   1     PROCESS:PROCEDURE(CMD);
249   2         DECLARE CMD BYTE;
250   2             DO I=0 TO T-1;
251   3                 DO J=0 TO T-1;
252   4                     OLDPTR=@OLDTAB(I*T+J);
253   4                     NEWPTR=@NEWTAB(I*T+J);
254   4                     NPTR=@NEWTAB((I+1 MOD (T-1))*T+J);
255   4                     EPTR=@NEWTAB(I*T+J+1 MOD (T-1));
256   4                     SPTR=@NEWTAB((I-1 MOD (T-1))*T+J);
257   4                     WPTR=@NEWTAB(I*T+J-1 MOD (T-1));
258   4                     DO CASE CMD;
259   5                         CALL INIT;  /* initialisation de NEWTAB */
260   5                         CALL CERCLE;/* trace de cercl  */
261   5                         CALL REMPLIR;   /* remplissag  */
262   5                         CALL AFF(REG);  /* affichage   un registre */
263   5                         ;
264   5                         CALL RAZ(REG,VAL);
265   5                         CALL SEGMENT;/*  trace de seg  nt*/
266   5                         END;
267   4                     END;
268   3                 END;
269   2         END PROCESS;

          $EJECT
```

PL/M-86 COMPILER ARRAY 15/11/83

```
 270   1      EXTREM:PROCEDURE;   /*initialise le trace d un segment*/
 271   2          DECLARE (XA,YA,XB,YB,XD,YD,XF,YF,DX,DY) BYTE;
 272   2          DECLARE (S1,VAL1,VAL2) BYTE;
 273   2          XD,XA=INVAL;YD,YA=INVAL;
 275   2          DX=(XB:=INVAL)-XA;XF=XB;
 277   2          DY=(YB:=INVAL)-YA;YF=YB;
 279   2          VAL1=2*DX;
 280   2          VAL2=2*DY;
 281   3          IF DX)127 THEN DO;DX=-DX;XD=XB;XF=XA;END;
 287   3          IF DY)127 THEN DO;DY=-DY;YD=YB;YF=YA;END;
 293   2          IF DY)DX THEN VAL=DY;
 295   2                  ELSE VAL=DX;
 296   2          DO I=XD TO XF;
 297   3              DO J=YD TO YF;
 298   4                  NEWPTR=@NEWTAB(J*T+I);
 299   4                  NPIXEL.R(2)=VAL1;
 300   4                  NPIXEL.R(3)=VAL2;
 301   4                  NPIXEL.R(4)=0;
 302   4                  NPIXEL.R(5)=VAL;
 303   4                  NPIXEL.ETAT=1;
 304   4                  END;
 305   3              END;
 306   2          NEWTAB(YA*T+XA).ETAT=2;
 307   2          NEWTAB(YB*T+XB).ETAT=2;
 308   2          END EXTREM;

             $EJECT
```

15/11/83

PL/M-86 COMPILER ARRAY

```
309   1     CENTRE: PROCEDURE; /*  initialise trace de cercle */
                /* R(2) = DX
                   R(3) = DY
                   R(4) = erreur =-R*R au centre; 0 sur le cercle
                   R(5) = valeur de test =2*R. */
310   2       DECLARE (XC,YC,RAY) BYTE;
311   2       XC=INVAL;YC=INVAL;RAY=INVAL;
314   2       DO I=XC-RAY TO XC+RAY;
315   3          DO J=YC-RAY TO YC+RAY;
316   4             NEWPTR=.NEWTAB(J*T+I);
317   4             NPIXEL.R(3)=(NPIXEL.X-XC)*4;
318   4             NPIXEL.R(2)=(NPIXEL.Y-YC)*4;
319   4             NPIXEL.R(4)=0;/*erreur nulle sur le cercle*/
320   4             NPIXEL.R(5)=4*RAY;
321   4             NPIXEL.ETAT=1;
322   4             END;
323   3          END;
324   2       NEWTAB((YC-RAY)*T+XC).ETAT=2;
325   2       NEWTAB((YC+RAY)*T+XC).ETAT=2;
326   2       NEWTAB(YC*T+XC-RAY).ETAT=2;
327   2       NEWTAB(YC*T+XC+RAY).ETAT=2;
328   1       END CENTRE;

        $EJECT
```

```
329  1    DO WHILE OFFH;
330  2        CALL CO(0DH);CALL CO(0AH);
332  2        CALL CO('?');CALL CO(0DH);CALL CO(0AH);
335  2        A=CI AND 7FH;CALL CO(A);
337  2        IF A='I' THEN CALL PROCESS(0);   /* initialisation */
339  2        IF A='C' THEN CALL PROCESS(1);   /* cercle */
341  2        IF A='B' THEN CALL CENTRE;       /* init cercle */
343  2        IF A='R' THEN CALL PROCESS(2);   /* remplissage */
345  2        IF A='D' THEN DO;                /* display registres
347  3                        REG=INREG;
348  3                        CALL PROCESS(3);
349  3                        END;
350  2        IF A='A' THEN CALL ACTIV;        /* activation cellules */
352  2        IF A='Z' THEN DO;                /* razz de registres */
354  3                        REG=INREG;
355  3                        VAL=INVAL;
356  3                        CALL PROCESS(5);
357  3                        END;
358  2        IF A='S' THEN CALL PROCESS(6);   /* tracé segments */
360  2        IF A='E' THEN CALL EXTREM;       /* initialise segments */
362  2        CALL RECOPIE;
363  2        END;

364  1    END ARRAY;
```

BIBLIOGRAPHIE

[BAJ 74], [BAT 80], [BOU 84], [BRE 65], [CAT 79], [CEA 82], [COD 68], [COU 76], [CRO 77], [DEG 77], [DUW 73], [FIK 83], [FOS 84], [FUP 81], [HAR 74], [HAR 81], [HUE 71], [JOR 74], [KID 83], [KUN 82], [LED], [MER 84], [NEU 66], [PAR 80], [PEL 84], [PLA 80], [POT 83], [PRE 83], [RED 84], [ROS 83], [STE 83], [TAN 83].

IV - MACHINE-OBJET

Comme nous l'avons indiqué au chapitre I, une façon d'augmenter le parallélisme est d'associer un processeur à chaque objet de la scène. Une étude complète est présentée dans ([DUR 83]) et nous ne faisons ici qu'en indiquer les grandes lignes.

IV.1 - PRINCIPES

IV.1.1 - Chaque processeur, associé à un objet, que nous appellerons processeur-objet, exécute tous les algorithmes nécessaires sur cet objet et produit finalement pour tout point (x, y) appartenant à l'objet la valeur des attributs à visualiser.

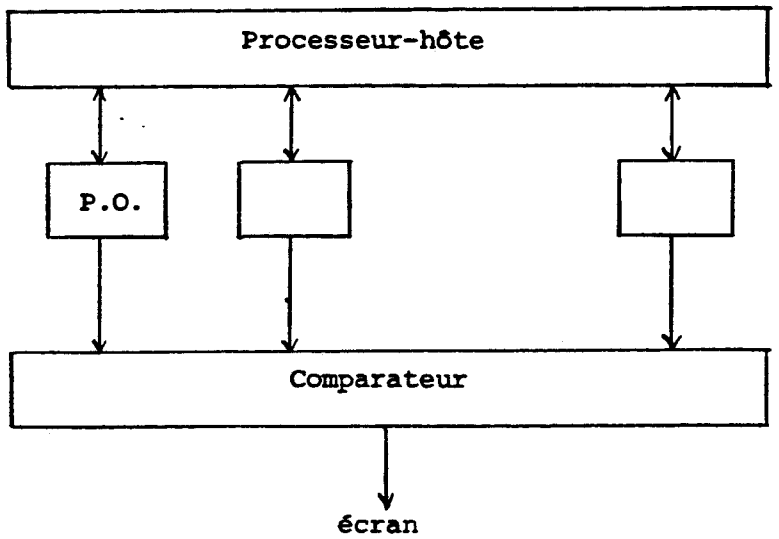
Ce que l'on cherche à réaliser pouvant être le temps réel, nous négligerons ici toutes les actions globales (transformations géométriques ou graphiques) pour nous attacher au stade ultime de la visualisation, à savoir

- le décodage de la description de la géométrie de l'objet
- l'évaluation des attributs à visualiser.

Parmi ces attributs figure z, la profondeur de l'objet au point considéré, qui nous permet de réaliser l'élimination des parties cachées, mais cela est une activité interobjets, donc interprocesseurs.

IV.1.2 - En résumé, une machine de synthèse par objets est composée de

- processeurs-objets évaluant à partir d'une description des objets l'ensemble des attributs à visualiser et en particulier la profondeur.
- un processeur global, que nous appellerons **comparateur**, qui détermine parmi les objets lequel (ou lesquels) est (sont) visible(s) en chaque point.



Architecture Générale

IV.1.3 - AVANTAGES ET INCONVIENTS

Parmi les inconvénients, on objectera immédiatement le grand nombre de processeurs-objets nécessaires pour obtenir une image complexe. A cela quatre réponses au moins :

. Cette approche n'est envisageable que dans le cadre d'une intégration poussée (VLSI), avec par exemple plusieurs processeurs par boîtier.

. On peut imaginer des processeurs physiques en faible nombre, alloués dynamiquement à des processeurs logiques.

. Beaucoup d'applications sont spécialisées, et demandent dans un premier temps peu d'objets, et donc peu de processeurs ; ceux-ci peuvent être également spécialisés et puissants (exemple : visualisation de molécules en chimie, à l'aide de processeurs de sphères).

. Une mémoire d'image de bonne définition contient entre 100 à 1500 boîtiers, qui n'existent plus dans cette architecture ; il est donc réaliste d'envisager une machine comportant 1000 processeurs-objets par exemple.

Au nombre des avantages, on peut citer évidemment :

. La modularité, c'est à dire l'indépendance des processeurs-objets vis-à-vis du reste du système: chacun d'eux peut être supprimé, remplacé sans difficulté.

. L'extensibilité, tant au niveau de la puissance de la machine (par adjonction de processeurs-objets supplémentaires) que de la gamme des applications (à l'aide de processeurs-objets spécialisés).

Citons en outre un aspect particulier, souvent réalisé de manière coûteuse : la facilité d'identification des objets ; en effet il y a correspondance quasi-directe entre les objets graphiques et les processeurs-objets.

Nous allons maintenant étudier trois machines, différant essentiellement par la manière dont le comparateur est réalisé.

IV.2 - MACHINE "MULTICRIBLE"

Cette machine dérive de la machine STYX, machine de synthèse d'images en deux dimensions et demi ([MER 79]).

IV.2.1 - CARACTERISTIQUES DE STYX

Pour ce qui nous concerne ici, nous rappellerons simplement ce qui suit :

a) Les objets élémentaires (rectangle, cercle, polygone) sont décodés par logiciel et leur contour est inscrit dans une mémoire intermédiaire appelée crible, contenant 1 bit par pixel.

b) Ces objets sont supposés dans un plan parallèle à l'écran. A chaque objet est associé un niveau z.

c) Un objet étant inscrit dans le crible, un processeur spécialisé l'inclut dans la mémoire de trame en temps réel de la manière suivante :

```
si (x, y) ∈ objet | alors si z < z mémoire de trame | alors MT = objet
                  |                                  | sinon MT = MT
                  | sinon MT = MT
```

On réalise ainsi de manière câblée un z-buffer.

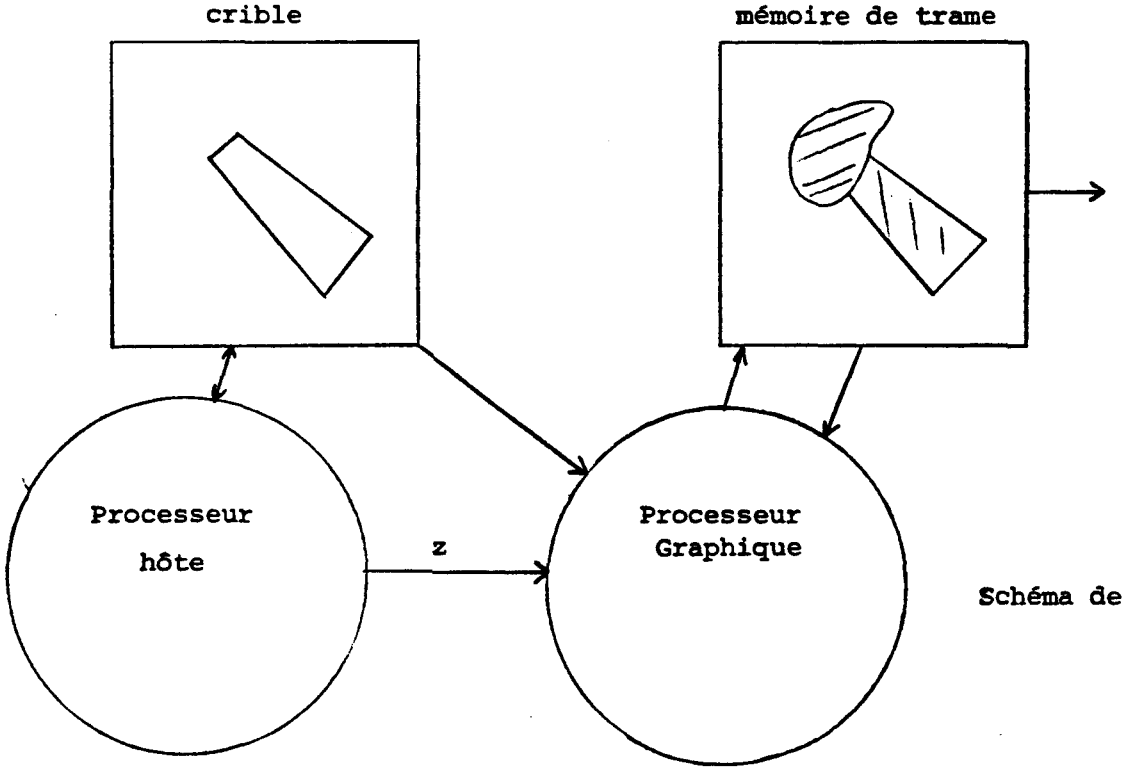


Schéma de principe STYX

IV.2.2 - GENERALISATION

Elle est immédiate : il suffit d'associer un crible à chaque objet, et de déterminer le niveau le plus proche de l'écran (z minimum) en tout point (x, y)

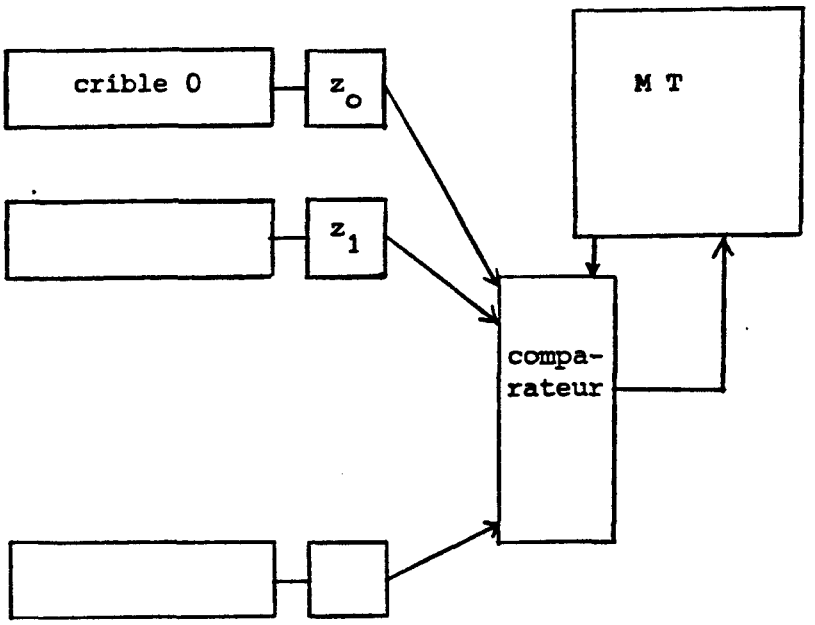


Schéma de principe de multi-STYX

Il est clair que, si cette machine à l'avantage d'être une extension simple d'une machine existante, elle a de nombreux inconvénients :

- deux dimensions 1/2 seulement
- décodage logiciel des objets (pas de temps réel)
- grosse quantité de mémoire pour les cribles (en 512×512 : 256 K bit/objet).

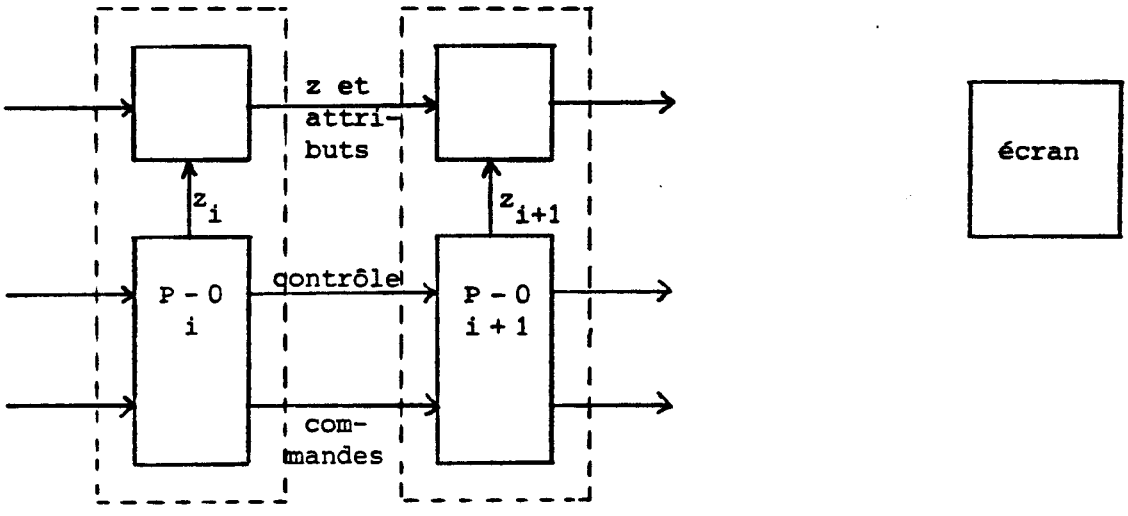
Néanmoins nous verrons en IV.4 qu'elle nous a conduit à l'étude d'une machine plus satisfaisante.

IV.3 - MACHINES PIPE-LINE

Deux machines-objets pipe-line ont été proposées presque simultanément en 1981 ([WEI 81],[ROK 81]). Nous retiendrons ici les caractéristiques principales de ces machines et leur faisabilité.

IV.3.1 - CARACTERISTIQUES COMMUNES

Chaque processeur-objet effectue la conversion de balayage pour un objet associé. Ces processeurs sont reliés en pipe-line et comportent un comparateur de z, suivant le schéma suivant :



A l'instant t, un étage reçoit

- (z, attributs) de son processeur voisin (de gauche)
- (z_i, $attributs_i$) du convertisseur de balayage associé.

Il élabore, en comparant z et z_i, un nouveau couple (z, attributs) qu'il transfère à son voisin (de droite).

Parmi les avantages d'une telle structure, citons :

. la régularité, l'uniformité, l'extensibilité, très favorables à une implémentation VLSI

. le parallélisme intrinsèque du pipe-line.

Mais citons quelques inconvénients :

. le nombre de fils sortant et entrant de chaque étage (notons qu'a priori l'accès par le processeur-hôte est aussi prévu par le pipe-line, ce qui implique de transférer beaucoup d'informations)

. le fonctionnement en temps différé : une modification d'un objet ne sera prise en compte qu'un certain temps plus tard, proportionnel à la longueur du pipe-line

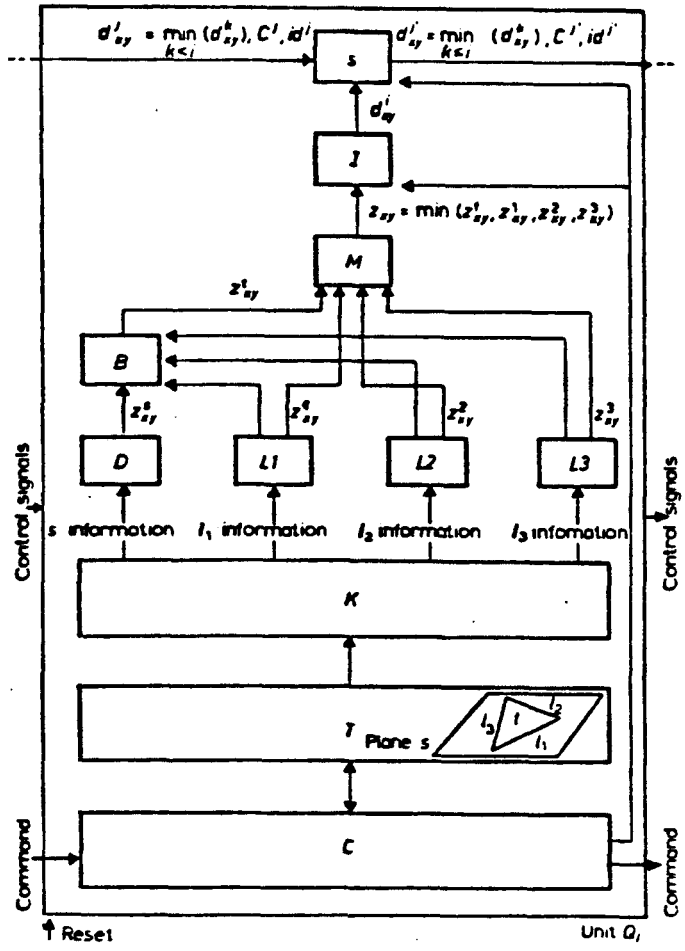
. la difficulté d'identifier un objet visible à partir d'un couple de coordonnées (x, y) de l'écran ; il faudrait transmettre les noms d'objets dans le pipe-line

. le problème de la tolérance aux pannes : un processeur peut bloquer toute la machine.

IV.3.2 - CARACTERISTIQUES SPECIFIQUES

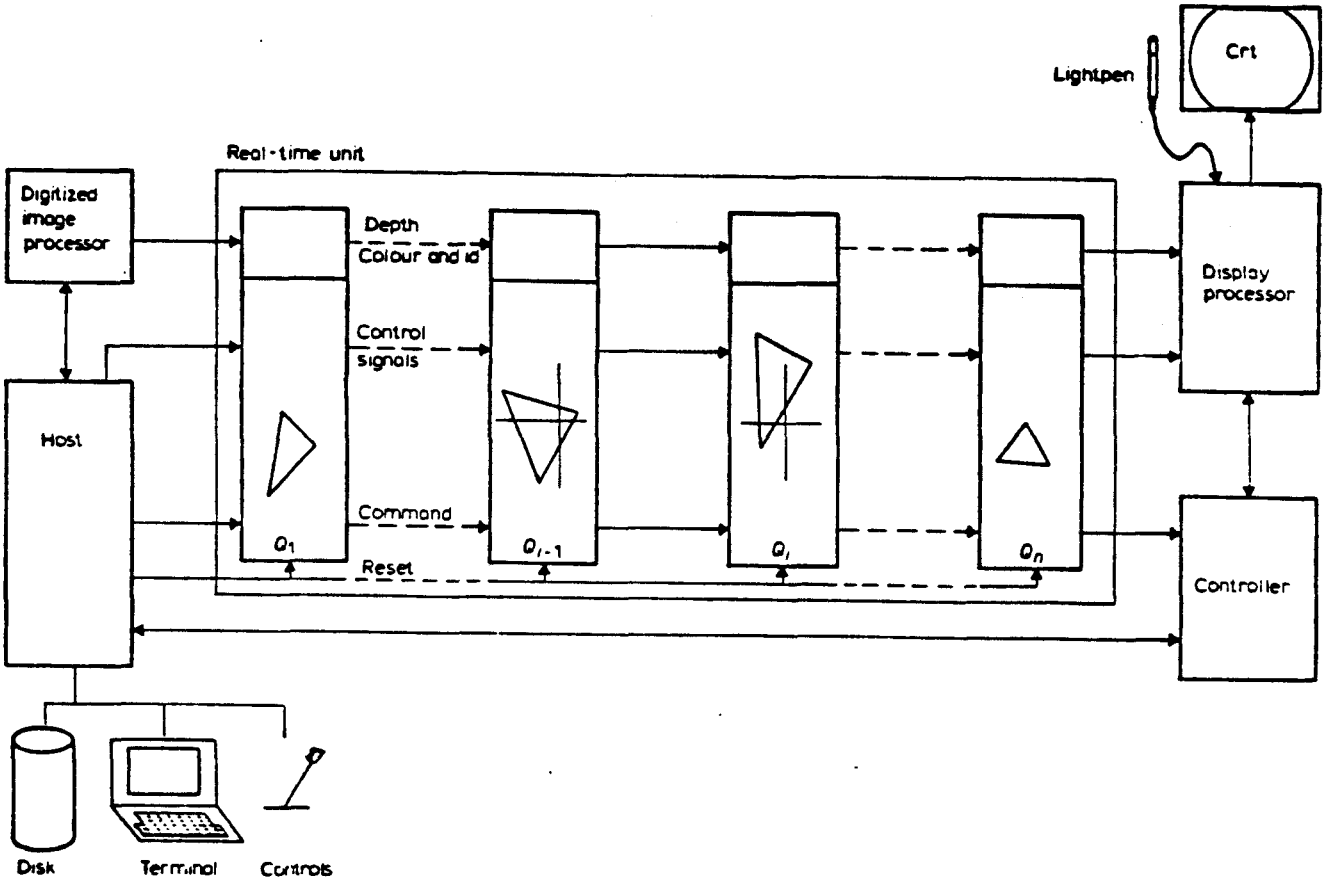
IV.3.2.1 - La machine de Roman et Kumura [ROK 81]

Cette machine ne traite que des triangles, un objet étant un ensemble de triangles. Elle autorise la pénétration d'objets.
L'accès par le processeur-hôte se fait par le pipe-line. Chaque processeur à l'architecture suivante :



La sous-unité C gère le transfert de commandes, le chargement et le vidage des triangles à partir du processeur-hôte.
La sous-unité T est supposée réaliser les transformations géométriques classiques (rotation, translation, mise à l'échelle, perspective). Les résultats sont transmis à K, générateur de constantes, qui élabore les valeurs nécessaires à la conversion de balayage en temps réel.
Les unités "temps réel", D, L_i, M, I élaborent la profondeur en (x, y), puis S détermine quelles valeurs (suivant z) doivent être transmises en sortie.

L'architecture complète du système est la suivante :



Notons pour terminer les problèmes suivants, soulevés par les auteurs :

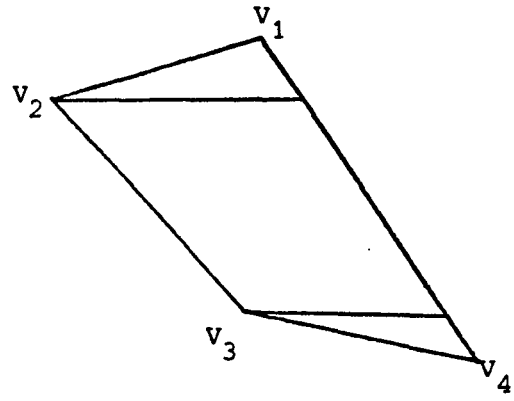
- la tolérance aux fautes
- le traitement des ombres portées
- le traitement de l'antialiasage
- le traitement correct de la transparence.

La machine a été simulée (ARTEMIS) et cette simulation, associée à des considérations technologiques, montre qu'elle est intégrable sous forme de VLSI.

IV.3.2.2 - La Machine de Weinberg

Elle traite des trapèzes plans dans l'espace, un objet étant un ensemble de trapézoïdes. De plus l'antialiasage est assuré par un fonctionnement en surrésolution (d'où problème de vitesse) suivi d'étages de filtrage 2×2. La machine traite partiellement la transparence. Les problèmes de l'accès par le processeur hôtes et de l'identification d'un objet ne sont pas abordés.

La machine a été simulée, et cette simulation montre, si besoin était, qu'elle est réalisable.



décomposition d'un polygone en trapèzes

IV.4 - LA MACHINE SANS MEMOIRE DE TRAME (SMT)

L'approche machine-objet, et en particulier l'étude des deux machines précédentes, nous a conduit à envisager une architecture que nous allons présenter maintenant.

IV.4.1 - OBJECTIFS - CONTRAINTES A RESPECTER

De ce qui précède, nous déduisons un certain nombre d'objectifs à atteindre pour cette machine :

1) Le désir d'intégration en VLSI impose de proposer une architecture modulaire, uniforme, régulière et extensible. Le nombre de fils de communication d'un module doit être faible.

2) Nous souhaitons réaliser une machine temps réel, qui nous permettra en particulier de nous affranchir de la mémoire de trame ; mais il faut aussi prévoir que cette machine objet devra être accessible facilement par le processeur hôte et autoriser des interactions rapides, notamment au niveau de l'identification et de la modification d'objets. Aussi choisissons nous une architecture **parallèle** et non pipe-line.

3) L'aspect tolérance aux fautes nous conforte dans cette approche : une machine parallèle peut fonctionner avec des processeurs en panne, ce qui est plus difficile pour une machine pipe-line.

4) Du point de vue strictement graphique, il est souhaitable de réaliser (par ordre d'importance)

- les surfaces cachées (cela va de soi), y compris en cas de pénétration
- l'antialisage
- le calcul d'attributs graphiques (couleur, texture, ombrage, lissage, mais aussi normales, etc ...)
- la transparence
- les transformations géométriques, les ombres portées.

5) En ce qui concerne les processeurs-objets, il faudrait pouvoir non seulement décoder des formes élémentaires simples, telles que segment, triangle ou trapèze, mais aussi des formes plus complexes (polygone, cercle, sphère, par exemple).

IV.4.2 - PRINCIPES

Nous ne faisons ici que rappeler quelques points que l'on trouvera très détaillés dans [DUR 83].

IV.4.2.1 - Structure générale

Comme indiqué au début de ce chapitre, cette machine comporte 3 parties principales :

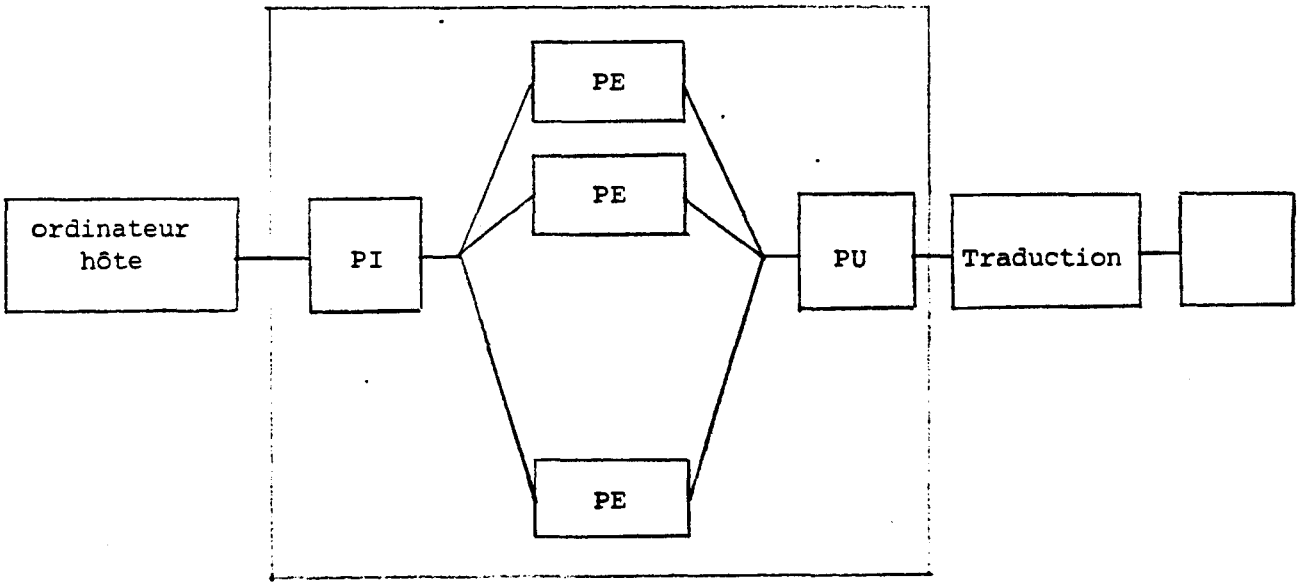
1) L'ensemble des processeurs-objets, appelés ici processeurs-éléments (PE).

2) Le processeur interface PI qui assure le dialogue

ordinateur <-> {PE}

3) Le processeur d'unification (PU), qui collecte les données fabriquées par les PE et sélectionne celles qui doivent être affichées.

Un mécanisme aval de traduction de ces données vers une information (couleur) acceptable par le support d'affichage complète cette architecture.



IV.4.2.2 - Les objets dans le système

Les objets graphiques de bases sont appelés "élément" ; un élément est associé à un PE. Nous avons choisi de typer les éléments, et donc les PE, de la manière suivante : un type correspond à une figure géométrique simple décrite en trois dimensions, interprétable par un PE. La suite de l'étude se limite aux types suivants : segment de droite, facette, polygonale, sphère ;

cette liste n'est pas exhaustive. Un élément sera qualifié par un certain nombre d'attributs, tels que son type (i.e. sa forme), sa taille, son emplacement, son aspect etc ...

Nous supposerons dans la suite qu'un processeur-élément ne peut traiter qu'un seul type d'élément.

L'ensemble des objets graphiques est généralement structuré, et nous choisissons ici une structuration arborescente : un objet peut être composé de sous-objets, eux-mêmes étant connus comme des objets. Les feuilles de l'arbre sont les éléments définis plus haut.

L'implémentation matérielle d'une telle arborescence impose

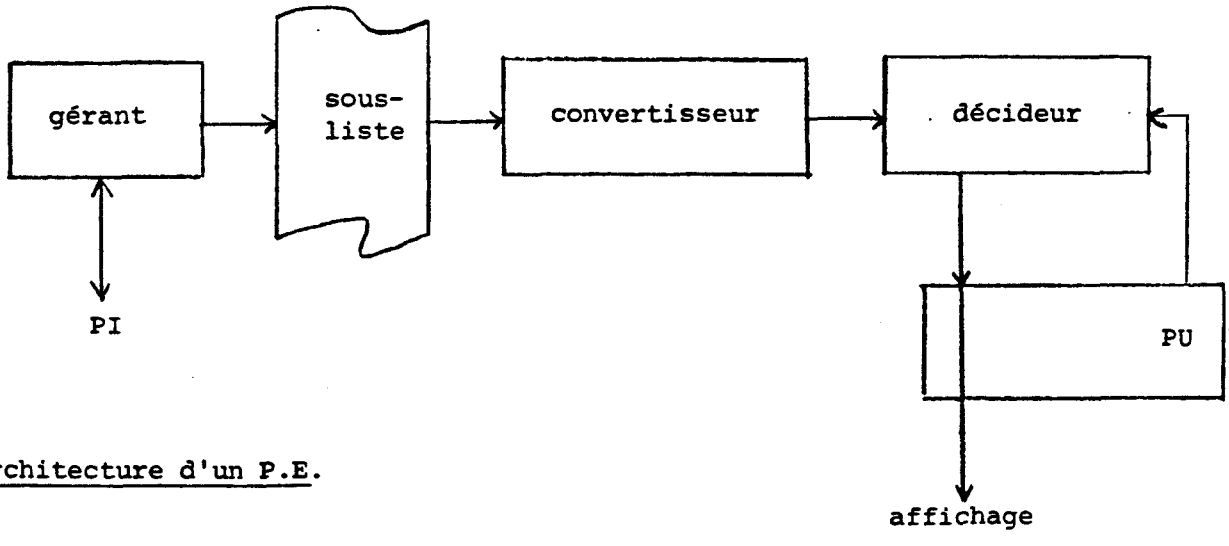
1) de figer l'arborescence, ou la sous-arborescence implémentée
2) d'obliger tous les éléments à se trouver au même niveau (le plus bas).

IV.4.2.3 - Les processeurs éléments

La principale fonction d'un processeur-élément est de convertir (en temps réel dans notre hypothèse) la morphologie et l'aspect d'un élément. Les résultats de conversion doivent être obtenus au rythme du balayage télévision, ligne par ligne et point par point. Les processeurs éléments contiennent la liste de visualisation du système graphique.

Chaque PE contient une sous-liste, accessible par le processeur interface PI, décrivant l'élément ; la partie `gérant' assure cette gestion de la sous-liste.

Chaque PE doit pouvoir également assurer le processus d'identification d'objet et également déterminer sa "visibilité" à l'aide de ce que lui fournit le Processeur d'Unification ; la partie "décideur" assure cette dernière fonction.



Architecture d'un P.E.

IV.4.2.4 - Le Processeur Interface

Le processeur interface a 2 fonctions principales

. gérer les communications avec l'extérieur (ordinateur hôte)

. gérer l'affectation des P.E. aux éléments.

IV.4.2.5 - Le Processeur d'Unification

Il effectue la synthèse des sorties des processeurs-éléments pour un pixel donné. Il résoud le problème des surfaces visibles en déterminant et sélectionnant l'élément visible en un point donné. S'il y a intersection de plusieurs éléments, il arbitre les conflits.

IV.4.3 - AVANTAGES ET INCONVENIENTS DE CETTE ARCHITECTURE

IV.4.3.1 - Interactions

Par rapport aux systèmes à mémoire de trame, il est clair que cette architecture facilite **l'identification** de tout objet repéré par les coordonnées d'un point (x, y) ; de même toute modification par le processeur-hôte d'un objet de la scène peut-être prise en compte "instantanément". Si le processeur-hôte est suffisamment rapide, (rappelons qu'il manipule les objets et non les "pixels") une animation est donc possible en temps réel.

IV.4.3.2 - Transparence

Il est ici utile de préciser que, la scène étant prête à la visualisation, il ne s'agit que de transparence dans l'axe des z, perpendiculairement à l'écran. Nous proposons une solution partielle à ce problème difficile dans le cadre suivant : nous faisons l'hypothèse qu'il n'y a qu'un seul objet transparent devant le premier objet visible opaque ; dans ce cas il suffit de choisir l'architecture à comparateur parallèle, en la modifiant légèrement. La sélection se fait sur tous les objets opaques ; cependant la comparaison de z se fait sur tous les objets, opaques ou non. On peut alors transférer en sortie par la voie normale les informations du premier objet opaque et par une voie supplémentaire (un bus par exemple) les informations de l'objet transparent éventuel. Il suffit alors de combiner ces deux informations.

IV.4.3.3 - Echantillonnage ("Anti-Aliasing")

Plusieurs solutions partielles peuvent être proposées :

. Un fonctionnement en sur-résolution, suivi d'un filtrage ; ceci implique que

chaque processeur fonctionne n^2 fois plus vite et pose donc des problèmes technologiques sérieux. Notons toutefois que le principe des processeurs est identique. Il suffit alors d'ajouter au système un tampon de n lignes et un filtre numérique temps réel, ce que l'on sait déjà réaliser sur les machines de traitement d'images.

. Un calcul plus précis (filtré) de T(i, j) par les processeurs-objets. La valeur élaborée T(i, j) tiendrait compte :

- . de l'occupation du pixel (sur les bords des objets)
- . de la taille des objets (pour les petits objets).

Dans ce cas, il faudrait tenir compte en un pixel donné de plusieurs objets, ce qui nous ramène au problème de la transparence.

IV.4.4 - IMPLEMENTATION ET EVALUATION

On trouvera dans ([DUR 83]) des résultats plus complets ; indiquons simplement qu'ont été réalisés en composants discrets :

- . un interpolateur de segment (Bresenham)
- . un processeur d'unification élémentaire.

D'autre part ont été simulés

- . d'autres interpolateurs de ligne droite
- . une machine complète comprenant des P.E. de types segments, polygones, sphères.

Il est clair à ce stade que seule une réalisation intégrée est envisageable.

CONCLUSION

Cette étude a mis en évidence à nouveau les diverses nécessités suivantes :

1) étude précise des algorithmes de base (interpolateurs)

2) intégration en VLSI ou CITGV.

Son originalité réside surtout dans la partie "Processeur d'Unification", dont la structure parallèle autorise l'identification instantanée d'objets sur l'image ; c'est de plus une structure de type bus permettant une extension simple de la machine.

BIBLIOGRAPHIE

[DUM 82], [DUR 83], [MER 79], [ROK 81], [WEI 81].

CONCLUSIONS GÉNÉRALES

Nous avions divers objectifs au début de ce travail : clarifier et classifier les différentes architectures d'ordinateurs susceptibles d'opérer sur des images ; remettre en cause certains préjugés quant aux algorithmes universels et revenir à l'étude précise des algorithmes de base ; montrer l'importance et la nécessité des composants à haute vitesse et à très haut degré d'intégration, et plus précisément d'architectures régulières construites à partir de composants tous identiques.

Si l'on peut comparer une machine-algorithme à une machine classique Von-Neumann, même perfectionnée (multiprocesseurs, récursivité, etc ...), il nous apparaît que les deux autres approches, les machines-objets (à pilotage par les données structurées de la scène), et les machines-pixels (à pilotage par les pixels de l'image finale), méritent des études plus approfondies. Parmi ces deux classes, ajoutons cependant que seule les machines pixels peuvent faire le lien entre traitement et synthèse d'images, et elles ont notre préférence pour des études ultérieures.

Cependant, divers points de ce travail, notamment le découpeur spatial, ont montré l'importance de la notion plus générale d'arbre quaternaire, qu'il conviendrait d'étudier plus précisément.

Aucune des machines proposées, ou d'autres machines dérivées de notre classification, n'est réalisable avec la technologie des portes Nand et du fer à souder : une réalisation effective, même si elle doit se baser sur des prototypes câblés et des simulations, nécessite la conception de composants VLSI ou CITGV : c'est l'évolution inéluctable de l'informatique et en particulier de l'informatique des images.

ANNEXES

ANNEXE 1 : ETUDE DU PARALLELISME

1.1 - Nous avons défini, et ce résultat est généralisable, que deux traitements peuvent être simultanés s'ils sont indépendants (conditions suffisantes). Nous supposons ici qu'un seul critère définit cette relation d'indépendance, qui, dans notre cas, pourra être par exemple

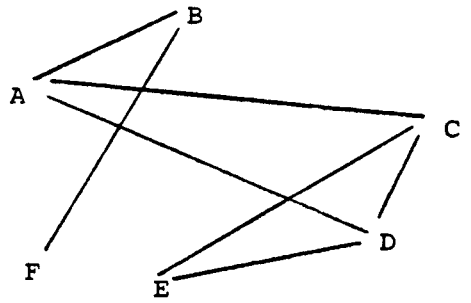
. un critère spatial lié à la géométrie de l'image (disjonction en y, en x et y, en z ...)

. un critère lié aux types des objets

. un critère temporel (pipe-line).

Nous cherchons donc à obtenir, parmi les traitements à réaliser, un minimum de groupes de traitements indépendants deux à deux.

Dans ce cas simple monocritère, la théorie des graphes nous apporte une solution optimale et une bonne heuristique :
En effet la relation que nous avons définie peut se représenter
- sous forme d'un graphe non orienté



sommet = traitement

arc = relation d'indépendance

- sous forme d'une matrice binaire symétrique

	A	B	C	D	E	F
A	X	1	1	1	0	0
B	1	X	0	0	0	1
C	1	0	X	1	1	0
D	1	0	1	X	1	0
E	0	0	1	1	X	0
F	0	1	0	0	0	X

Nous proposons dans la suite une méthode pour obtenir une "bonne" partition de ce graphe en sous graphes ne comportant que des traitements indépendants deux à deux.

Ainsi, dans le petit exemple ci-dessus, on pourrait obtenir :

(ACD) et (BF) et (E)

ou (ECD) et (AB) et F

ou (ECD) et (A) et (BF)

1.2 - Compte tenu de la méconnaissance courante de certaines propriétés des graphes, il nous a paru utile de rappeler ici un certain nombre de définitions et propriétés des graphes.

1. Recouvrement

Une famille d'ensembles $(A_i)_{i\in I}$ est un recouvrement de l'ensemble E si elle possède les propriétés suivantes :

$$\bigcup_{i\in I} A_i = E \quad \text{et} \quad A_i \neq \emptyset \ \forall\, i \in I$$

2. Partition

Une famille d'ensembles $(A_i)_{i\in I}$ est une partition de l'ensemble E si :

- c'est un recouvrement de E

et si - $\forall\, i \in I, \forall\, j \in I \mid i \neq j \quad A_i \cap A_j = \emptyset$

Propriété : étant donné un recouvrement de E, on peut toujours en déduire par intersections et différences une partition de E.

3. Degré

On appelle degré d(A) d'un sommet A d'un graphe symétrique ou non orienté le nombre d'arêtes incidentes à A.

4. Clique

Une clique est un préordre complet symétrique : chaque sommet du sous-ensemble ainsi formé est relié à chacun des autres.

5. Coloration

Soit G = (X, U) un graphe et Y un ensemble (de couleurs). Une coloration du graphe G est une application

$$\gamma : X \to Y$$

telle que deux sommets adjacents dans G ont des images distinctes dans Y.

L'ensemble des sommets de même couleur engendre un sous graphe dépourvu d'arcs.
Le nombre minimum de classes susceptible de constituer une telle partition de X est appelé "un nombre chromatique" $\gamma(G)$.

6. Ensemble stable (intérieurement)

Un sous-ensemble I du graphe G = (X, V) est un e.s.i. si deux sommets quelconques lui appartenant ne sont pas adjacents. Un tel ensemble est maximum si il n'existe pas d'autre e.s.i. le contenant strictement. Le nombre maximum de sommets de G susceptibles de former un e.s.i. est appelé nombre de stabilité interne, noté $\alpha(G)$.

Propriétés

1) L'ensemble des e.s.i. de $\bar{G}$ est égal à l'ensemble des cliques de G

2) Les cliques maximales de G sont les esim de $\bar{G}$, le nombre maximum de sommets d'une clique est noté $\theta(G)$.

Autres propriétés

Toute coloration des sommets correspond à une partition en e.s.i.

7. Définition : Nombre cliquomatique

C'est le nombre minimum de cliques nécessaires pour former une partition des sommets de G. Ce nombre, noté $\chi(G)$, est donc égal à $\gamma(\bar{G})$.

Théorèmes :

$$\begin{cases} \alpha(G) \cdot \gamma(G) \geq n(G) \\ \theta(G) \cdot \chi(G) \geq n(G) \end{cases}$$

$$\begin{cases} \alpha(G) \leq \chi(G) \\ \theta(G) \leq \gamma(G) \end{cases}$$

avec α = taille max d'un e.s.i
θ = taille max d'une clique
γ = nombre min d'e.s.i. formant une partition (nombre chromatique)
χ = nombre min. de cliques formant une partition (nombre cliquomatique)
n = nombre de sommets

8. Ensemble stable extérieurement

Un sous-ensemble I du graphe G = (X, V) est un e.s.e. si tout sommet $x \notin I$ est adjacent à au moins un élément de I.

Propriété :

Dans un graphe symétrique, tout e.s.i.m est un e.s.e.
Le nombre minimum de sommets susceptibles de former un e.s.e s'appelle $\beta(G)$, nombre de stabilité externe.

1.3 - Le problème se pose donc en termes de théorie des graphes de la façon suivante :
Etant donné un graphe $G = (X, U)$ symétrique, réaliser une partition comportant un minimum de parties à l'aide de cliques.
Ce problème peut se remener à la recherche d'un recouvrement (minimal) des sommets du graphe par des cliques maximales. Il s'agit en réalité de colorer le graphe $\bar{G}$ en un nombre γ de classes.

Il existe une solution idéale ([ROY 70]) consistant essentiellement à énumérer l'ensemble des cliques maximales de G, i.e. les e.s.i.m. de $\bar{G}$, à l'aide d'une méthode arborescente, puis à chercher un nombre minimum d'e.s.i.m. formant un recouvrement de $\bar{G}$, d'où l'on déduit une partition de $\bar{G}$, donc de G, comportant exactement $\gamma(G)$ éléments.

Mais cette méthode, arborescente devient impraticable dès que n est grand. On appliquera donc plutôt le théorème de Pavell et Welsh suivant :

Les n sommets d'un graphe G étant numérotés $i = 1, \ldots n$ suivant l'ordre décroissant de leur degré d_i, soit k le numéro du dernier sommet vérifiant

$$k \leq d_k + 1$$

on a $\gamma(G) \leq k$

(Corollaire = $\gamma(G) \leq d_n + 1$)

On en déduit l'algorithme suivant, qui produit une coloration utilisant au plus k couleurs :
α) Numéroter les sommets du graphe conformément à l'ordre des degrés décroissants.
β) Sélectionner le sommet de plus petit indice, soit x_{i1} ; sélectionner ensuite, parmi les sommets non adjacents à x_{i1} celui de plus petit indice, soit x_{i2} ; idem parmi les sommets non adjacents à x_{i1} et x_{i2} ; lorsqu'il n'existe plus de sommets, l'ensemble des x_{ij} trouvés forme une classe coloriable
γ) Supprimer du graphe les sommets ainsi coloriés ; aller en β s'il reste des sommets.

Exemple :

On a le graphe symétrique suivant, que l'on cherche à partitionner

	a	b	c	d	e	f	g	h	i	j
a	1	0	1	1	0	0	1	0	0	0
b	0	1	1	0	1	0	0	1	1	1
c	1	1	1	0	0	1	0	1	1	1
d	1	0	0	1	0	0	1	0	0	1
e	0	1	0	0	1	1	1	1	1	1
f	0	0	1	0	1	1	0	0	1	1
g	1	0	0	1	1	0	1	0	0	1
h	0	1	1	0	1	0	0	1	0	0
i	0	1	1	0	1	1	0	0	1	1
j	0	1	1	1	1	1	1	0	1	1

étape α :

		a	b	c	d	e	f	g	h	i	j
nombre de zéro	= degré	6	4	3	6	3	5	5	6	4	2
	numéro	1	6	8	2	9	4	5	3	7	10

↑

Le théorème de Pavell et Welsh indique que $\gamma(\bar{G}) = \theta(G) = 5$

Le graphe $\bar{G}$ est donc réorganisé de la manière suivante

	a	d	h	f	g	b	i	c	e	j	couleur
a 1	0	0	1	1	0	1	1	0	1	1	1
d 2	0	0	1	1	0	1	1	1	1	0	1
h 3	1	1	0	1	1	0	1	0	0	1	2
f 4	1	1	1	0	1	1	0	0	0	0	3
g 5	0	0	1	1	0	1	1	1	0	0	1
b 6	1	1	0	1	1	0	0	0	0	0	2
i 7	1	1	1	0	1	0	0	0	0	0	3
c 8	0	1	0	0	1	0	0	0	1	0	2
e 9	1	1	0	0	0	0	0	1	0	0	3
j 10	1	0	1	0	0	0	0	0	0	0	3

étape β :

1) on choisit a
   puis ad
   puis adg
   on supprime (adg)

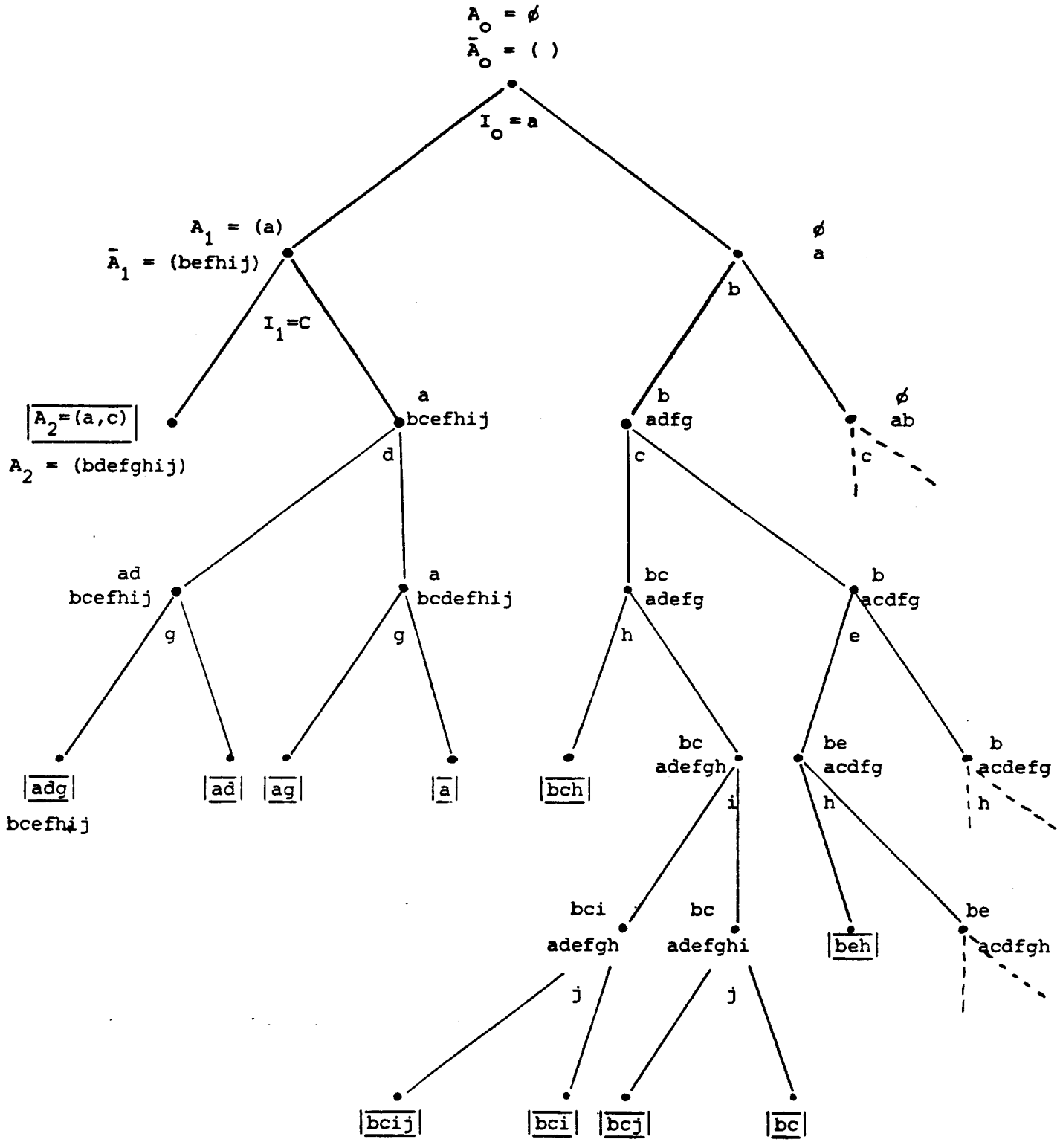
2) on choisit h
   puis hb
   puis hbc
   puis (?) hbce ne convient pas (à cause de c-e)
   on supprime (hbc)

3) on choisit f
   puis fi
   puis fie
   puis fiej
   on supprime (fiej)

on obtient donc une partition de G en 3 cliques, au lieu des 5 (maximum) prévues,

1 adg
2 hbc
3 fiej

Si, sur le même exemple on effectue la méthode systématique, arborescente, on obtient :


$A_o = \phi$
$\bar{A}_o = ( )$
$I_o = a$
$A_1 = (a)$
$\bar{A}_1 = (befhij)$
$I_1 = C$
$\overline{|A_2=(a,c)|}$
$A_2 = (bdefghij)$
a
bcefhij
d
ad
bcefhij
g
a
bcdefhij
g
$\overline{|adg|}$
bcefhij
$\overline{|ad|}$
$\overline{|ag|}$
$\overline{|a|}$
b
adfg
c
ϕ
a
b
ϕ
ab
c
bc
adefg
h
b
acdfg
e
$\overline{|bch|}$
bc
adefgh
i
be
acdfg
h
b
acdefg
h
bci
adefgh
j
bc
adefghi
j
$\overline{|beh|}$
be
acdfgh
$\overline{|bcij|}$
$\overline{|bci|}$
$\overline{|bcj|}$
$\overline{|bc|}$

Recensement des cliques de l'exemple

$\boxed{ac}$	$\boxed{adg}$ *	
ad		
ag		
bc	$\boxed{bch}$ *	
be	bci) $\boxed{bcij}$	
bh	bcj)	
bi	$\boxed{beh}$	
bj	bei) $\boxed{beij}$	
	bej)	
	bij	
cf		
$\boxed{ch}$		
ci	cfi) $\boxed{cfij}$	
cj	cfj)	
dg	$\boxed{dgj}$	
dj		
ef		
eg	efi) $\boxed{efij}$ *	
$\boxed{eh}$	efj)	
ei	$\boxed{egj}$	
ej	eij	
fi	fij	
fj		
gj		
ij		

* résultat trouvé

$\boxed{\quad}$ cliques maximales

d'où $\theta(G) = 4$

$\chi \geq n/\theta = \frac{10}{4} \Rightarrow \chi \geq 3$

en réalité l'algorithme donne $\begin{cases} \chi \leq 3 \\ \theta(G) \leq 5 \end{cases}$

cliques maximales	P1	P2
ac		
adg	X	X
bch	X	
beh		X
bcij		
beij		
cfij		X
dgj		
egi		
efij	X	

P1 est la partition trouvée

P2 est une autre partition possible

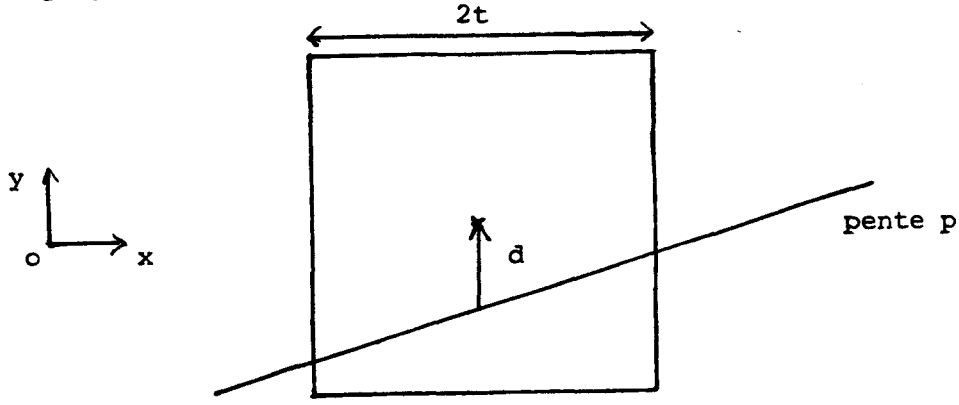
Une étude plus précise du coût en temps et espace de cet algorithme reste à effectuer. En tous cas, il a le mérite de produire un résultat rigoureux et justifié, ce qui n'est pas si fréquent en parallélisme.

ANNEXE 2 : ANTIALIASSAGE DE SEGMENTS ET DE CERCLES

Nous montrons ici comment l'on peut réaliser l'antialiassage de segments et de cercles à partir de l'erreur calculée en chaque pixel entre la position du pixel et la position idéale du contour ; cette erreur est généralement calculée car elle est la base des algorithmes incrémentaux.

2.1 - TRACE DE SEGMENT/POLYGONE

Soit un pixel de côté 2t (= 1 en unité pixel) et un segment idéal de pente p, passant à une distance d du centre du pixel.

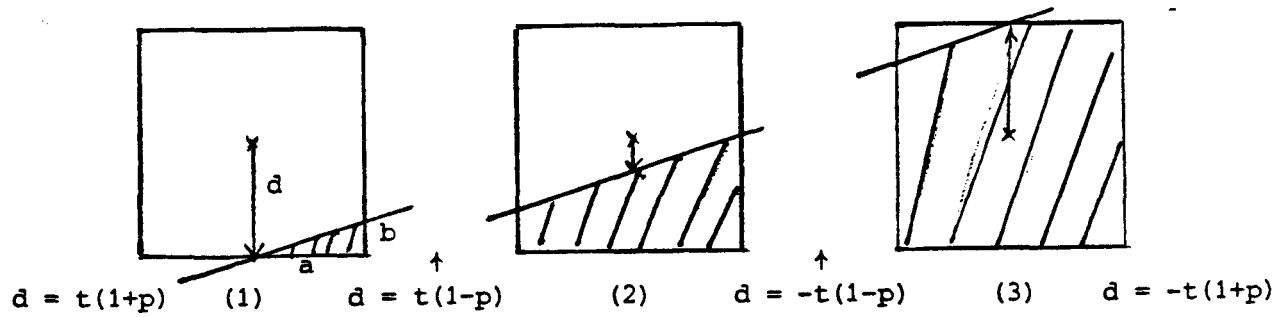


Nous appliquons ici la méthode des "surfaces" : on cherche à évaluer la portion du pixel occupée par le tracé. Nous faisons donc ici les hypothèses suivantes :

1) $p \leq 1$
2) $-t(1+p) \leq d \leq t(1+p)$
3) le tracé est soit a) un segment d'épaisseur "verticale" ≥ 1
   soit b) un polygone

limités supérieurement par le segment idéal indiqué.

Nous obtenons donc les cas suivants



cas 1 : il est facile de montrer que

$$\begin{cases} a = t + \frac{t-d}{p} \\ b = a \times p = t-d+tp \end{cases}$$

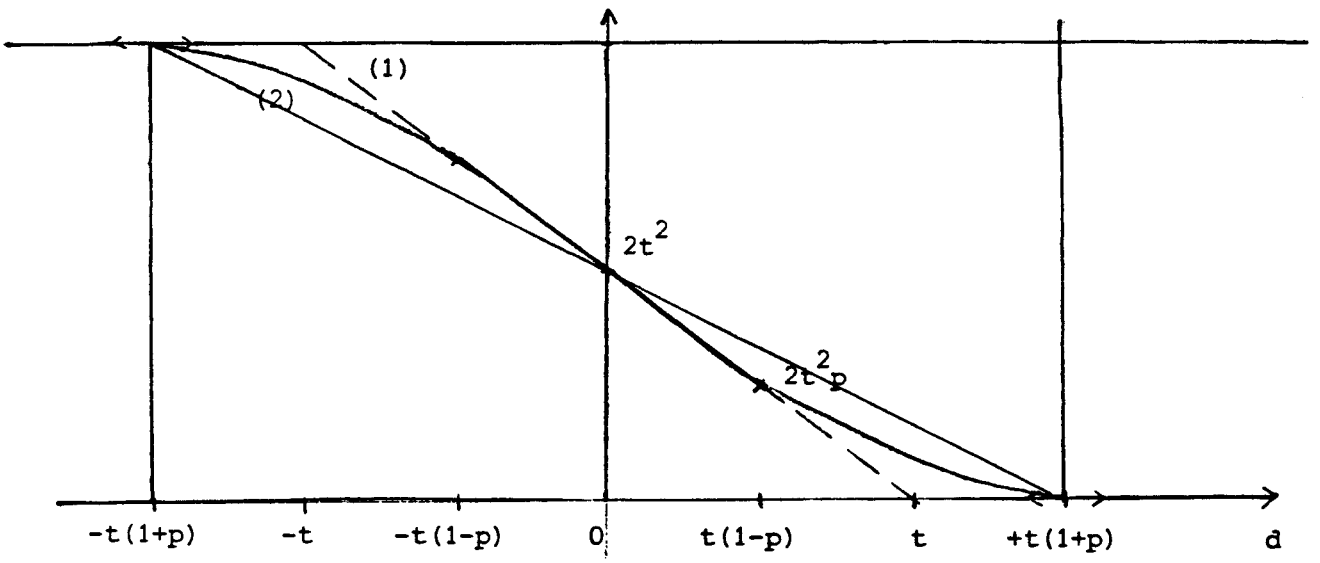
d'où $\boxed{2s_1 = \left(t + \frac{t-d}{p}\right)^2 \times p} = t^2 \frac{(p+1)^2}{p} - 2td \frac{p+1}{p} + \frac{d^2}{p}$

cas 2 : $\boxed{s_2 = 2t(t-d)} = 2t^2 - 2td$

cas 3 : analogue au cas 1

$$\boxed{2s_3 = 8^{t^2} - \left(t + \frac{t+d}{p}\right)^2 \times p}$$

d'où le diagramme récapitulatif suivant : (tracé avec $p = \frac{1}{2}$)



D'où deux approximations limites (1) et (2).

→ D_1 : $s = -2td + 2t^2 = \frac{1}{2} - d$ en unités-pixels

l'erreur maximale commise est

$\varepsilon_1(d=t) = \frac{1}{2} t^2 p = p/8$ en unités-pixels

[Note : pour $-t < d < +t$ seulement]

$\rightarrow D_2 : s = \frac{-2t}{1+p} d + 2t^2$

l'erreur maximale commise est

$\varepsilon_2 \left(d = t \frac{p^2+1}{p+1}\right) = 2t^2 \frac{p}{(p+1)^2} = \frac{p}{2(p+1)^2}$ en unité-pixels.

[Note : on a donc toujours $\varepsilon_2 \geq \varepsilon_1$]

Une bonne approximation est donc l'expression simple

$$\boxed{s = \frac{1}{2} - d}$$

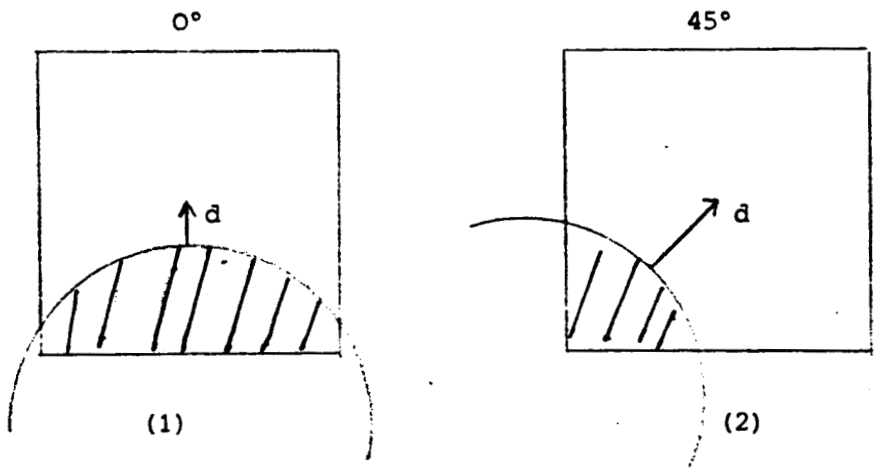
à appliquer pour $\boxed{-\frac{1}{2} < d < +\frac{1}{2}}$

Remarque

Des calculs, plus complexes, peuvent également être réalisés avec un pixel circulaire de rayon t' ; moyennant le choix, fort justifié, de $t' > t$ ($t' \simeq \sqrt{\frac{\mu}{\pi}}\, t$), les résultats obtenus sont analogues, et l'approximation proposée ci-dessus reste valable.

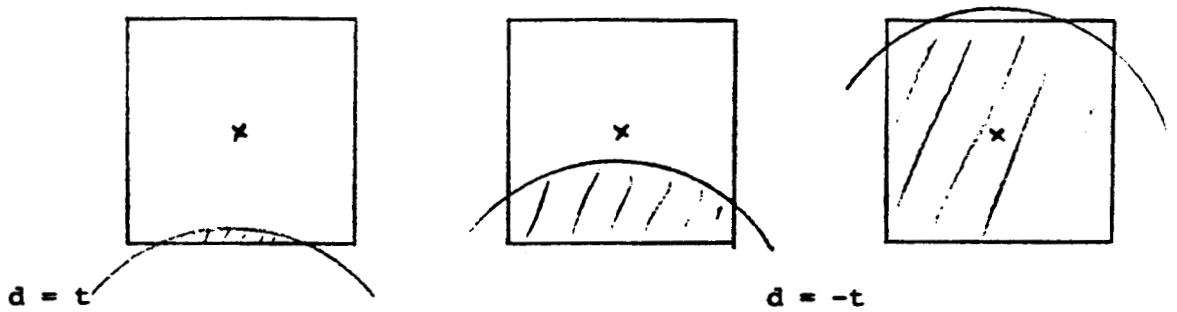
2.2 - TRACE DE CERCLE (DISQUE)

Les algorithmes usuels de tracé de cercle évaluent la puissance du pixel par rapport au cercle idéal. Cette puissance est $P = d^2 - R^2$. Nous effectuerons les calculs sur les deux cas simples suivants, les autres cas supposant une simulation numérique.

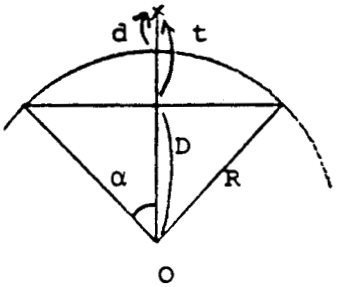


2.2.1 - Cas 0°

Nous avons les trois cas suivants :



Evaluons d'abord la surface du secteur circulaire



$$\begin{cases} D = R \cos \alpha \\ D = R - (t-d) \end{cases}$$

$$S = R^2(\alpha - \sin \alpha \cos \alpha)$$

La formule exacte est :

$$S = R^2\left[\arccos\left(1 - \frac{t-d}{R}\right) - \left(1 - \frac{t-d}{R}\right)\sqrt{1-\left(1 - \frac{t-d}{R}\right)^2}\ \right]$$

Si l'on admet que α est faible, ce qui sera le cas si $R >> t$, c'est à dire dans la plupart des cas, on aura

$$\begin{cases} \sin \alpha \simeq \alpha \\ \cos \alpha \simeq 1 - \frac{\alpha^2}{2} \Rightarrow \alpha^2 \simeq 2 \times \left(1 - \frac{D}{R}\right) = 2\left(\frac{t-d}{R}\right) \end{cases}$$

d'où $$\boxed{S \simeq R^2 \frac{\alpha^3}{2} \simeq \sqrt{2R(t-d)^3}}$$

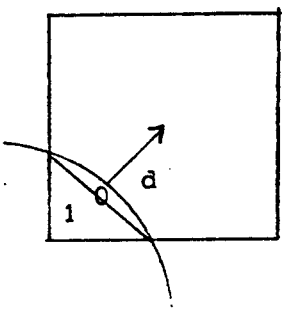
α est maximal si R sin α = t ou $\alpha \simeq \frac{t}{R}$

d'où $\boxed{S_{max} \simeq \frac{t^3}{2R}} = \frac{1}{16R}$ unités-pixels

La surface du secteur circulaire est donc négligeable et le problème se ramène au segment horizontal

$$\boxed{s = \frac{1}{2} - d}$$

2.2.2 - Cas 45°



$S = S_o + S_1$

* S_o est évalué comme précédemment, sauf que t devient $t' = t\sqrt{2}$ d'où

$$S_o \simeq \sqrt{2R(t\ 2\text{-}d)^3} \simeq R^2 \frac{\alpha^3}{2}$$

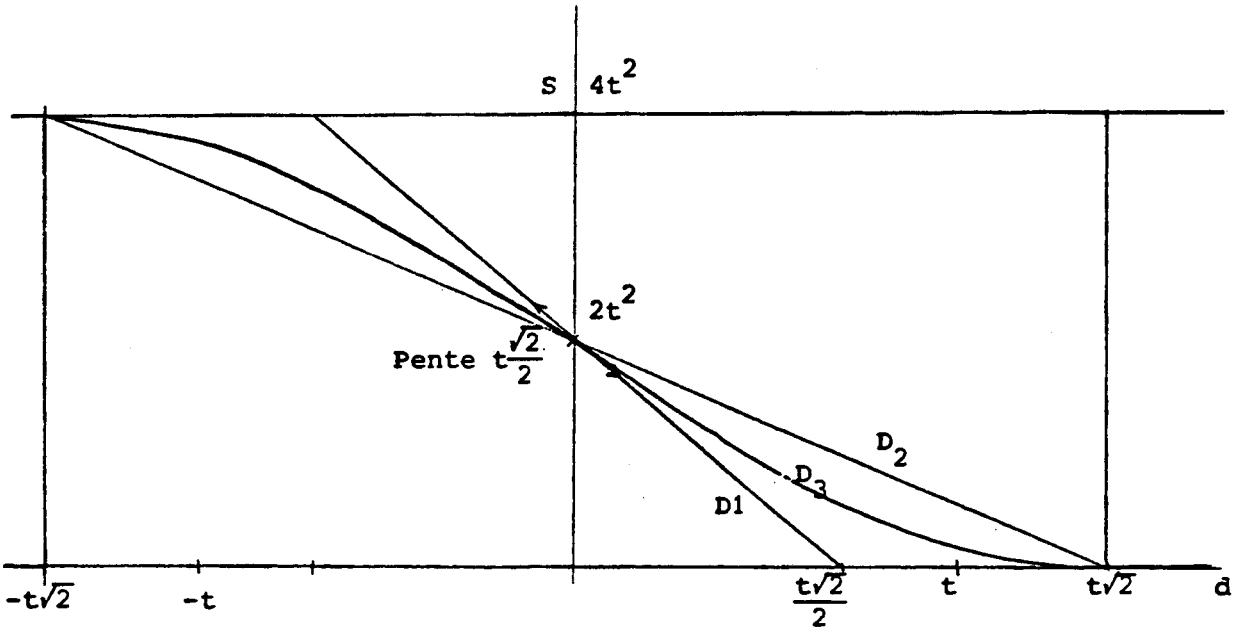
α est maximale si R sin $\alpha = t\sqrt{2}$ ou $\alpha \simeq \frac{t\sqrt{2}}{R}$

d'où S_o max $\simeq \frac{t^3\sqrt{2}}{R} = \frac{\sqrt{2}}{8R}$ unités pixels

On négligera donc S_o.

* $S_1 \simeq (t\sqrt{2} - d)^2$ si $d \geq 0$

d'où le diagramme



On admettra donc que $S = (t\sqrt{2}-d)^2$.

On peut approximer ces paraboles par des droites :

$$D_1 : S = -\frac{t\sqrt{2}}{2} d + 2t^2$$

$$D_2 : S = -t\sqrt{2} + 2t^2$$

ou $D_3 : S = -2td + 2t^2$ qui est l'approximation de segment.

Nous étendrons donc notre formule $\boxed{S = \frac{1}{2} - d}$ au traitement de cercle, moyennant une erreur commise au second ordre, que nous avons montrée négligeable dès que R est grand.

CONCLUSION

Les calculs menés ont pu paraître longs et fastidieux pour un résultat aussi simple ; mais ils sont nécessaires pour justifier cette approximation, que l'on utilise généralement intuitivement. Le résultat simple S = 1/2 - d pourra donc être utilisé sans arrière-pensée et sans calculs supplémentaires, la valeur de d étant fournie par les algorithmes classiques de tracé.

BIBLIOGRAPHIE

[ACQ 82] ACQUAH J. and al
A conceptual Model of Raster Graphics Systems.
Computer Graphics, Vol 16, N° 3, July 1982.

[ALL 83] ALLAIN G.
Images de synthèse dans les simulateurs de vol.
Bulletin INRIA, n° 88, 1983.

[ALL 84] ALLAIN G.
Images de synthèse dans les simulateurs d'entraînement.
Colloque Image, Biarritz, Mai 84.

[ANP 76] ANDREWS H.C., PATTERSON C.L.
Digital Interpolation of Discrete Images.
IEEE Transactions on Computers, Vol C.25, n° 2, Fév. 76.

[ATH 77] ATHERTON P. and al.
Polygon Shadow Generation.
Computer Graphics, Vol 11, n° 2, 1977.

[BAJ 74] BARRETT R.C., JORDAN B.W.
Scan Conversion Algorithms for a Cell Organized Raster Display.
Communications ACM, Vol 17, n° 3, March 74.

[BAR 74] BARNHILL R.E., RIESENFELD R.F.
Computer Aided Geometric Design.
Academic Press, 1974.

[BAS 83] BASSETT S.
Powerful custom VLSI chips drive graphics machines.
Computer design, December 1983, p. 39.

[BAS 84] BASILLE J.L. et al.
Structures parallèles en traitement d'images.
Colloque Image, Biarritz, Mai 84.

[BAT 80] BATCHER K.E.
Design of a Massively Parallel Processor.
IEEE Trans. Computers, Vol C-29, n° 9, Sept. 80.

[BER 63] BERGE
La théorie des graphes et ses applications.
Dunod, 1963.

[BER 77] BERTIN J.
La graphique et le traitement graphique de l'information.
Flammarion, 1977.

[BLN 76] BLINN J.F., NENELL M.E.
Texture and Reflexion in Computer Generated Images.
Communications ACM, Vol 19, n° 10, Oct. 76.

[BLI 77a] BLINN J.F.
Models of light reflexion for computer synthesized pictures.
Computer Graphics, siggraph ACM, Vol 11, n° 2, 1977.

[BLI 77b] BLINN J.F.
Simulation of wrinkled surfaces.
Computer Graphics, Vol 11, n° 2, 1977.

[BON 82] BONO P.R. et al.
GKS - The First Graphics Standard.
IEEE Computer Graphics, July 82. p 9.

[BOU 70] BOUKNIGHT W.J.
A procedure for generation of three dimensional half-toned computer graphics representations.
Communications ACM, vol 13, n° 9, sept. 70.

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

[BOU 84] BOUVILLE C. et al.
La synthèse d'images par lancer de rayons : algorithmes et architectures.
Colloque Image, Biarritz, Mai 84.

[BRA 81] BRADIER A.
Eléments pour la représentation de données graphiques dans un contexte interactif.
Mémoire de DEA, Lille, 1981.

[BRD 81] BRIGHAM R.C., DUTTON R.D.
Graphs which, with their complements, have certain clique covering numbers.
Discrete Mathematics 34, p. 1, 1981.

[BRE 65] BRESENHAM J.E.
Algorithm for computer control of a digital plotter.
IBM Systems J., Vol 4 n° 1, 1965.

[BRK 73] BRON C. and KERBOSCH J.
Algorithm 457 : Finding All Cliques of an Undirected Graph.
Communication ACM, Vol 16, n° 9, Sept. 73.

[BUI 75] BUI TUONG PHONG
Illumination for Computer Generated Images.
Communications ACM, Vol 18, n° 6, June 1975.

[CAR 76] CARLSON E.D. and al.
Multiple Colors and image mixing in graphics terminals.
IBM Research Division, RJ 1874 (27023) 11/22/76.

[CAT 79] CATMULL E.
A Hidden-Surface Algorithm with Anti-Aliasing.
Computer Graphics, 1979, p. 6.

[CEA 82] CEA, INRIA, EDF
La réalisation de logiciels graphiques interactifs.
Eyrolles, 1982.

[CLA 80] CLARK J.
A VLSI Geometry Processor for Graphics.
Computer, July 80, p. 59.

[CLA 82] CLARK J.
The Geometry Engine : a VLSI Geometry System for Graphics.
Computer Graphics, Vol 16, n° 3, July 82.

[COD 68] CODD E.F.
Cellular Automata
Academic Press, New-York, 1968.

[CON 79] CONNOR P.
A pratical approach to colour graphic computer terminals.
Eurographics 79, p. 274.

[CON 80] CONRAC
Raster Graphics Handbook.
1980.

[COU 76] COUSIN R.
Etude et définition d'un processeur modulaire de communication à haut niveau de parallèlisme.
Thèse de Doctorat de Spécialité, Lille, 1976.

[CRO 77] CROW F.C.
The Aliasing Problem in Computer-Generated Shaded Images.
Communications of the ACM, Nov 77, Vol 20, n° 11.

[CRO 78] CROW F.C.
Shaded Computer Graphics in the Entertainment Industry.
Computer, March 78, p. 11.

[DAL 81] DANIELSON E., LEVIALDI S.
Computer Architecture for Pictorial Information System.
Computer, Novembre 81.

[DAN 70] DANIELSSON P.E.
Incremental Curve Generation.
IEEE Trans. on Computers, C. 19, n° 9.

[DAN 84] DANIELSSON P.E.
Algorithms and architectures for image processing.
Colloque Image, Biarritz, Mai 84.

[DEG 77] DEGRENDEL B.
Communications et protocole dans une architecture multi-micro-processeurs.
Thèse de Docteur-Ingénieur, Lille, 1977.

[DOR 83] DOU M. REGNIER T.
La synthèse d'images par ordinateur en France : une nécessaire synergie.
COM'7, n° 22, p. 27.

[DUM 82] DURIF Ph. MERIAUX M.
Une architecture pour la synthèse d'images sans mémoire de trame.
Congrès AFCET 82, Lille.

[DUR 83] DURIF Ph.
Etude d'une Machine Parallèle de Synthèse d'Images à Découpage par Objets.
Thèse de 3e cycle, Lille, 1983.

[DUW 73] DUFF M.J.B., WATSON D.M., et al.
A cellular Logic Array for Image Processing.
Pattern Recognition, Vol 5, p. 229, 1973.

[EFC] EFCIS
Notices Techniques EF 9365/66/67.

[ENC 75] ENCARNACAO J.
Computer Graphics : Programmierung und Anwendung von graphischen Systemen.
R. Oldenburg Verlag, Münich, 1975.

[FER 81] FERREIRA N.
Conception et Réalisation d'un système interactif pour la synthèse d'images réalistes : HELIOS
Thèse de Docteur-Ingénieur, Grenoble, 81.

[FIK 83] FISHER A.L., KUNG H.T.
Synchronizing Large VLSI Processor Arrays.
ISCA 83, p. 54.

[FOS 84] FOUQUES M., SAMY R.
Algorithmes de traitement d'images et réseaux systoliques.
Colloque Image, Biarritz, Mai 84.

[FOV 82] FOLEY J.D., VAN DAM A.
Fundamentals of Interactive Computer Graphics.
Addison Wesley, 82.

[FRE 61] FREEMAN H.
On the encoding of arbitrary geometric configurations.
IRE Transactions on Computers, Vol 10, n° 2, June 61.

[FRS 75] FREEMAN H., SHAPIRA R.
Determining the minimum-area encasing rectangle for an arbitrary closed curve.
Communications ACM, Vol 18, n° 7, July 75.

[FRW 82] FRANKLIN M.A., WANN D.F.
Asynchronous and clocked control structures for VLSI Based interconnection Networks.
IEEE, ISCA 82.

[FUC 77] FUCHS H.
Distributing a Visible Surface Algorithm Over Multiple Processors.
Proc. ACM 77, October 77, Seattle.

[FUP 81] FUCHS H., POULTON J.
PIXEL-PLANES : A VLSI-oriented Design for Raster Graphics Engine.
VLSI Design, Vol 2, N°3, 1981, p. 20.

[FUR 76] FU K.S., ROSENFELD A.
Pattern Recognition and Image Processing.
IEEE Trans. Computers, Vol C-25, n° 12, Dec. 76

[GAI 84] GAILLAT G.
Le calculateur parallèle Capitan : 600 Mips pour l'imagerie temps réel.
Colloque Image, Biarritz, Mai 84.

[GAM 69] GALIMBERTI R., MONTANARI U.
An algorithm for hidden-line elimination
Communications ACM, Vol 12, n° 4, April 69.

[GAV 77] GAVRIL F.
Algorithms on clique separable graphs.
Discrete Mathematics 19, p. 159, 1977.

[GOU 71] GOURAUD H.
Computer Display of Curved Surfaces.
University of UTAH, UTEC CSc 71113, June 71.

[GRA 71] GRAY S.B.
Local properties of binary images in two dimensions.
IEEE Trans. Computers, Vol C-20, n° 5, May 71.

[GRA 80] GRAVE M.
Etude d'un noyau de système de synthèse d'images. Application à la visualisation de scènes tridimensionnelles.
Thèse de Docteur-Ingénieur, Lille, Décembre 80.

[GRI 75] GRIFFITHS J.G.
A Data Structure for the Elimination of Hidden Surfaces by Patch Subdivision.
Computer Journal, Vol 7, n° 3, July 75.

[GRR 79] GRIMONPREZ G., RAUCH A.
Présentation du Logiciel STYX
Journées AFCET-TTI, mai 1979.

[GUS 81] GUPTA S. and al.
A VLSI Architecture for Updating Raster-Scan Displays.
Computer Graphics, Vol 15, n° 3, Aout 81.

[HAR 74] HARDOIN-DUPARC J.
Paradis terrestre dans l'automate cellulaire de Conway.
RAIRO, R-3, 1974, p. 63.

[HAR 81] HARDOUIN-DUPARC J.
Autocell
Rapport Greco n° 9, Bordeaux, 1981.

[HUE 71] HUECKEL M.H.
An operator which Locates Edges in Digitized Pictures.
Journal ACM, Vol 18, n° 1, Janvier 71.

[HWA 83] HWANG K.
Computer Architectures for Image Processing.
Computer, Jan. 83, p. 10.

[JO 80] JOURNAL OFFICIEL
Terminologie.
11 décembre 1980.

[JOB 73] JORDAN B.W., BARRETT R.C.
A Scan conversion algorithm with reduced storage requirements.
Communications ACM, Vol 16, n° 11.

[JOR 73] JORDAN B.W. and al.
An improved algorithm for the generation of nonparametric curves.
IEEE Trans. on Computers, Dec. 73.

[JOR 74] JORDAN B.W., BARRETT R.C.
A Cell Organized Raster Display for line Drawings.
Communications ACM, Vol 17, n° 2.

[KAI 83] KAI HWANG, KING-SUN FU
Integrated Computer Architectures for Image Processing and Database Management.
Computer, Jan. 83, p. 51.

[KID 83] KIDODE M.
Image Processing Machines in Japan.
Computer, Jan. 83, p. 68.

[KIL 81] KILGOUR A.C.
A hierarchical model of a graphics system.
Computer Graphics, April 1981.

[KUN 82a] KUNG S.Y. and al.
Wavefront Array Processor : Language, Architecture, and Applications.
IEEE Transactions on Computers, Vol C-31, n° 11, Nov 82.

[KUN 82b] KUNG H.T.
Lecture Notes for Advance Course on VLSI Architecture.
Bristol, 19-30 July 1982.

[LAC 80] LANE J.M. and al.
Scan Line Methods for Displaying Parametrically Defined Surfaces.
Communications de l'ACM, Janvier 80, Vol 23, n° 1.

[LED] LEDRU A.
Algorithmes de Remplissage pour la Synthèse d'images.
Rapport Interne, IMAG, Grenoble.

[LER 78] LERAY P.
Génération en temps réel d'images synthétiques tridimensionnelles.
Journée d'Etude INRIA - 26 avril 78.

[LER 80] LERAY P.
Réalisation d'un système de Génération Synthétique d'images en temps réel.
Congrès AFCET, Novembre 80.

[LEW 83] LEWIS D.
Use of VLSI in Computer Graphics.
Séminaire "Synthèse d'Images", Octobre 1983, INRIA.

[LOC 80] LOCEFF
The line.
Computer, June 80, p. 56.

[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, Dec. 77.

[LUC 78] LUCAS M.
Quelques voies de recherches dans le domaine de la communication graphique avec un ordinateur.
AFCET 1978.

[LUC 81] LUCAS M.
Survol des techniques de réalisation de logiciels pour des surfaces à pointillage.
Journées d'étude AFCET-TTI, Avril 81.

[LUC 84] LUCAS M.
La synthèse d'Images par Ordinateur.
Colloque Image, Biarritz, Mai 84.

[MAR 76] MARTELLI A.
An application of Heuristic Search Methods to Edge and Contour Detection.
Communications ACM, Vol 19, n° 2.

[MAR 77] MARTINEZ F.
Etude des problèmes de conception et de réalisation d'animation : le système SAFRAN.
Thèse 3e cycle, Grenoble, Mai 77.

[MAR 79] MARTINEZ F.
An approach to the modelling and display of landscapes.
Eurographics 79, p. 301.

[MAR 81] MARTINEZ F., FERREIRA N.
HELIOS, terminal interactif pour la synthèse d'images réalistes.
Congrès AFCET, Novembre 81.

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

[MAR 84] MARTINEZ F.
Vers une approche banalisée des synthétiseurs d'image.
Colloque Image, Biarritz, Mai 84.

[MAT 78] MATHERAT P.
Conception d'un circuit intégré pour la visualisation graphique.
Thèse de 3e cycle, Paris VI, 1978.

[MEC 80] MEAD C.A., CONWAY L.
Introduction to VLSI Systems.
Addison-Wesley, 1980.

[MER 79] MERIAUX M.
Etude et réalisation d'un terminal graphique couleur tridimensionnel fonctionnant par taches.
Thèse de Docteur-Ingénieur - Lille, 79.

[MER 81] MERIAUX M.
Architectures adaptées à la synthèse d'images.
Congrès AFCET, Nov 81, Cif/Yvette.

[MER 82] MERIAUX M.
La Programmation des Couleurs en Informatique Graphique.
Publication Interne n° 22, Déc. 1982, Lille.

[MER 84a] MERIAUX M.
A Two-Dimensional Clipping Divider.
Eurographics 84, Copenhague, Sept. 84.

[MER 84b] MERIAUX M.
Architectures de Machines d'Images : une approche méthodique, quelques exemples.
Colloque Image, Biarritz, Mai 84.

[MER 84c] MERIAUX M.
A Cellular Architecture for Image Synthesis
Microprocessing and Microprogramming Vol 13, p. 179, 1984.

[MOL 76] MORVAN P., LUCAS M.
Images et ordinateur : introduction à l'infographie interactive.
Larousse, 1976.

[MYE 80] MYERS W.
Computer Graphics : a two-way street.
Computer, July 80, p. 49

[MYE 81] MYERS W.
Computer Graphics : Reaching the User.
Computer, March 81, p. 7.

[NEC] NEC Electronics
Notices techniques µPD7220/GDC.

[NEG 77] NEGROPONTE N.
Raster scan approaches to computer graphics.
Computer & Graphics, Vol 2, p. 179.

[NEN 72] NEWEL M.E., NEWELL R.G., SANCHA T.L.
A solution to the hidden-surface problem.
Proceedings of the ACM annuel conference, Boston 1972, Vol 1, p 443.

[NES 79] NEWMAN W. et SPROULL R.
Principles of Interactive Computer Graphics.
Mc Graw Hill, New York, 1979.

[NEU 66] NEUMANN Von J.
Theory of Self Reproducing Automata.
University of Illinois Press, Urbana, 1966.

[NEW 76] NEWMAN W.M.
Trends in Graphics Display Design.
IEEE Trans. on Computers, Vol C-25, n° 12, Dec 76.

[NIS 83] NISHIMURA H. and al.
Links 1 : A parallel pipelined multimicrocomputer system for image creation.
ISCA 83, p. 387.

[NOL 71] NOLL A.M.
Scanned-Display Computer Graphics.
Communication ACM, vol 14, n° 3, March 71.

[OLE 82] OLEJNIK R.
Réalisme dans les images générées par ordinateur.
Mémoire de DEA, Lille, 1982.

[PAR 80] PARKE F.I.
Simulation and expected performance Analysis of multiple processor z-buffer systems.
Computer Graphics, Siggraph 80 p. 48.

[PAR 82] PARKER R.
Input/Output Technology.
Computer Design, Dec 82, p. 157.

[PAU 82] PAULIDIS T.
Algorithms for Graphics and Image Processing.
Springer Verlag, New York, 1982.

[PEL 84] PELERIN M.
Algorithmes graphiques parallèles.
Colloque Image, Biarritz, Mai 84.

[PET 83] PETITPREZ B.
Modèle Arborescent de Description de Logiciels Interactifs facilitant les Reprises.
Thèse de 3e cycle - Lille, Sept. 83.

[PLA 80] PLANCKE P.
Etude des réseaux cellulaires de micro-processeurs.
Thèse de Docteur-Ingénieur, Lille, 1980.

[POT 83] POTTER J.L.
Image Processing on the Massively Parallel Processor.
Computer, Jan 83, p. 62.

[PRE 83] PRESTON K.
Cellular Logic Computers for Pattern Recognition.
Computer, Jan. 83, p. 36.

[RAN 82] RANNOU R.
Tris pour machines synchrones.
Rapport de Recherches n° 154 - IRISA - Aout 82.

[REC 83] LA RECHERCHE
La révolution des images.
n° 144. Mai 1983.

[RED 84] REDJIMI M.
Réalisation d'un système parallèle pour le graphique.
Colloque Image, Biarritz, Mai 84.

[REN 83] RENNES 1983, Journées d'Etudes
Nouvelles Images Electroniques.

[RIC 82] RICE R.
VLSI Tritorial.
IEEE Computer Society Press, 1982.

[ROK 81] ROMAN G.C., KIMURA T.
VLSI perspective of real-time hidden-surface elimination.
VLSI, Vol 13, n° 2, March 81.

[ROS 83] ROSENFELD A.
Parallel Image Processing Using Cellular Arrays.
Computer, Jan 83, p. 14.

[ROY 70] ROY B.
Algèbre moderne et théorie des graphes.
Dunod 1970.

[SCH 73] SCHNEIDER M.
Méthodes pour recenser toutes les cliques maximales et θ-maximales d'un graphe.
RAIRO, Vol 3, sept 73, p. 21.

[SHA 79] SHAW A.
Basic Algorithms and Programming Languages.
Ecole d'Eté CEA - INRIA - EDF, Juillet 79.

[SKL 72] SKLANSKY J. and al.
Minimum Perimeter Polygons of Digitized Silhouettes.
IEEE Trans. Computers, Vol C-21, n° 3, March 72.

[SKL 76] SKLANSKY J. and al.
Parallel Detection of Concavities in Cellular Blobs.
IEEE Trans. Computers, Vol C-25, n° 2, Feb. 76.

[SMI 78] SMITH A.R.
Color Gamut Transform Space.
Proceedings Siggraph'78, p. 12.

[SMI 79] SMITH A.R.
Tint Fill
Computer Graphics, Vol 13, n° 3, p. 276, 1979.

[SPS 68] SPROULL R.F., SUTHERLAND I.E.
A Clipping Divider
AFIPS Conf. Proc. Vol 33, 1968, p. 765.

[STE 83] STERNBERG S.R.
Biomedical Image Processing.
Computer, Jan. 83, p. 22.

[STT 79] STAUDHAMMER J., TARTAR J.
Shaded Graphics Hardware.
Eurographics 79, p. 68.

[SUH 74] SUTHERLAND I.E., HODGMAN G.W.
Reentrant Polygon Clipping.
Communications of the ACM, Vol 17, n° 1, p. 32.

[SUM 74] SUZUKI Y. and al.
Computer driven color graphic display device with color painting.
Compcon 74, Feb. 74.

[SUS 74] SUTHERLAND I.E., SPROULL R.F., SCHUMACKER R.G.
A characterization of ten hidden-surface algorithms.
ACM computing surveys, vol 6, n° 1, mars 74.

[TAN 83] TANIMOTO S.L.
A Pyramidal Approach to Parallel Processing.
ISCA 83, p. 372.

[WAR 69] WARNOCK J.E.
A hidden surface algorithm for computer generated halftone pictures.
University of Utah, Dept. of Comp. Science, 1969.

[WAT 75] WATKINS G.S.
A Real-Time Visible surface algorithm.
University of Utah, Dept. of Comp. Science, 75.

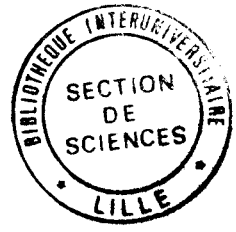
[WEA 77] WEILER K., ATHERTON P.
Hidden surface removal using polygon area sorting.
Computer Graphics, Vol 11, n° 2, 1977.

[WEI 81] WEINBERG R.
Parallel Processing Image Synthesis and Antialiasing.
Computer Graphics, Vol 15 n° 3, Aout 1981.

[WHE 82] WHELAN D.S.
A Rectangular Area Filling Display System Architecture.
Computer Graphics, Vol 16, n° 3, July 82.

[WIS 81] WISE J.L., SZEJNWALD H.
Display Controller simplifies design of sophisticated graphics terminals.
Electronics, April 7, 1981, p. 153.

[ZYD 82] ZYDA M.J.
A contour display generation algorithm for VLSI implementation.
Computer Graphics, Vol 16, n° 3, July 82.



RÉSUMÉ

Le traitement et la synthèse des images par ordinateur fait intervenir une grande quantité de traitements sur un grand nombre de données ; l'étude des entités mises en jeu nous conduit à définir trois classes d'architectures de machines spécialisées dans l'informatique des images. Ces trois classes sont étudiées de divers points de vue : parallélisme, intégration, performances, etc... puis sont chacune explicitées sur un exemple de machine.

Ainsi est indiqué un exemple particulier de "machine-algorithme", à savoir un découpeur bidimensionnel, cellule de base de nombreux problèmes de synthèse et de traitement, de même est étudiée en détail une "machine-pixel", où l'on associerait un processeur élémentaire à chaque pixel de l'image ; enfin on donne les grandes lignes d'une "machine-objet" à structure parallèle.

Ce travail tend à montrer d'une part l'intérêt et la faisabilité de machines et de composants spécialisés en traitement et/ou synthèse d'images, d'autre par la pauvreté des algorithmes usuels appliqués à de telles architectures et la nécessité d'études précises et fondamentales dont nous avons donné quelques exemples.

MOTS-CLÉS

imagerie informatique, synthèse d'images, traitement d'images, architectures spécialisées.