USTL
FLANDRES ARTOIS

LIFL
U.A. 369 du C.N.R.S.

LABORATOIRE D'INFORMATIQUE FONDAMENTALE DE LILLE

Année 1991

Numéro d'ordre: 721

THESE

présentée à

l'Université des Sciences et Techniques de LILLE FLANDRES ARTOIS

Pour Obtenir le titre de

Docteur en Informatique

Titre:

le projet PASTIS,

reconnaissance approchée

de dessins au trait

Par Jérôme DELEU

soutenue le lundi 13 Mai 1991 devant la commission d'Examen

Membres du Jury:

Jean FRANÇON	Président
Irène ZAMBETTAKIS	Rapporteur
Max DAUCHET	Rapporteur
Michel MERIAUX	Directeur de Thèse
Alain SALMON	Examinateur

UNIVERSITE DES SCIENCES
ET TECHNIQUES DE LILLE
FLANDRES ARTOIS

DOYENS HONORAIRES DE L'ANCIENNE FACULTE DES SCIENCES

M.H. LEFEBVRE, M. PARREAU.

PROFESSEURS HONORAIRES DES ANCIENNES FACULTES DE DROIT ET SCIENCES ECONOMIQUES, DES SCIENCES ET DES LETTRES

MM. ARNOULT, BONTE, BROCHARD, CHAPPELON, CHAUDRON, CORDONNIER, DECUYPER, DEHEUVELS, DEHORS, DION, FAUVEL, FLEURY, GERMAIN, GLACET, GONTIER, KOURGANOFF, LAMOTTE, LASSERRE, LELONG, LHOMME, LIEBAERT, MARTINOT-LAGARDE, MAZET, MICHEL, PEREZ, ROIG, ROSEAU, ROUELLE, SCHILTZ, SAVARD, ZAMANSKI, Mes BEAUJEU, LELONG.

PROFESSEUR EMERITE

M. A. LEBRUN

ANCIENS PRESIDENTS DE L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE

MM. M. PAREAU, J. LOMBARD, M. MIGEON, J. CORTOIS.

PRESIDENT DE L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE FLANDRES ARTOIS

M. A. DUBRULLE.

PROFESSEURS - CLASSE EXCEPTIONNELLE

M. CONSTANT Eugène	Electronique
M. FOURET René	Physique du solide
M. GABILLARD Robert	Electronique
M. MONTREUIL Jean	Biochimie
M. PARREAU Michel	Analyse
M. TRIDOT Gabriel	Chimie Appliquée

PROFESSEURS - 1ère CLASSE

M. BACCHUS Pierre	Astronomie
M. BIAYS Pierre	Géographie
M. BILLARD Jean	Physique du Solide
M. BOILLY Bénoni	Biologie
M. BONNELLE Jean-Pierre	Chimie-Physique
M. BOSCQ Denis	Probabilitéss
M. BOUGHON Pierre	Algèbre
M. BOURIQUET Robert	Biologie Végétale
M. BREZINSKI Claude	Analyse Numérique

M. BRIDOUX Michel	Chimie-Physique
M. CELET Paul	Géologie Générale
M. CHAMLEY Hervé	Géotechnique
M. COEURE Gérard	Analyse
M. CORDONNIER Vincent	Informatique
M. DAUCHET Max	Informatique
M. DEBOURSE Jean-Pierre	Gestion des Entreprises
M. DHAINAUT André	Biologie Animale
M. DOUKHAN Jean-Claude	Physique du Solide
M. DYMENT Arthur	Mécanique
M. ESCAIG Bertrand	Physique du Solide
M. FAURE Robert	Mécanique
M. FOCT Jacques	Métallurgie
M. FRONTIER Serge	Ecologie Numérique
M. GRANELLE Jean-Jacques	Sciences Economiques
M. GRUSON Laurent	Algèbre
M. GUILLAUME Jean	Microbiologie
M. HECTOR Joseph	Géométrie
M. LABLACHE-COMBIER Alain	Chimie Organique
M. LACOSTE Louis	Biologie Végétale
M. LAVEINE Jean-Pierre	Paléontologie
M. LEHMANN Daniel	Géométrie
Mme LENOBLE Jacqueline	Physique Atomique et Moléculaire
M. LEROY Jean-Marie	Spectrochimie
M. LHOMME Jean	Chimie Organique Biologique
M. LOMBARD Jacques	Sociologie
M. LOUCHEUX Claude	Chimie Physique
M. LUCQUIN Michel	Chimie Physique
M. MACKE Bruno	Physique Moléculaire et Rayonnements Atmosph.
M. MIGEON Michel	E.U.D.I.L.
M. PAQUET Jacques	Géologie Générale
M. PETIT Francis	Chimie Organique
M. POUZET Pierre	Modélisation - calcul Scientifique
M. PROUVOST Jean	Minéralogie
M. RACZY Ladislas	Electronique
M. SALMER Georges	Electronique
M. SCHAMPS Joel	Spectroscopie Moléculaire
M. SEGUIER Guy	Electrotechnique
M. SIMON Michel	Sociologie
Melle SPIK Geneviève	Biochimie
M. STANKIEWICZ François	Sciences Economiques
M. TILLIEU Jacques	Physique Théorique
M. TOULOTTE Jean-Marc	Automatique
M. VIDAL Pierre	Automatique
M. ZEYTOUNIAN Radyadour	Mécanique

PROFESSEURS - 2ème CLASSE

M. ALLAMANDO Etienne	Composants Electroniques
M. ANDRIES Jean-Claude	Biologie des organismes
M. ANTOINE Philippe	Analyse
M. BART André	Biologie animale
M. BASSERY Louis	Génie des Procédés et Réactions Chimiques

Mme BATTIAU Yvonne	Géographie
M. BEGUIN Paul	Mécanique
M. BELLET Jean	Physique Atomique et Moléculaire
M. BERTRAND Hugues	Sciences Economiques et Sociales
M. BERZIN Robert	Analyse
M.BKOUCHE Rudolphe	Algèbre
M. BODARD Marcel	Biologie Végétale
M. BOIS Pierre	Mécanique
M. BOISSIER Daniel	Génie Civil
M. BOIVIN Jean-Claude	Spectroscopie
M. BOUQUELET Stéphane	Biologie Appliquée aux enzymes
M. BOUQUIN Henri	Gestion
M. BRASSELET Jean-Paul	Géométrie et Topologie
M. BRUYELLE Pierre	Géographie
M. CAPURON Alfred	Biologie Animale
M. CATTEAU Jean-Pierre	Chimie Organique
M. CAYATTE Jean-Louis	Sciences Economiques
M. CHAPOTON Alain	Electronique
M. CHARET Pierre	Biochimie Structurale
M. CHIVE Maurice	Composants Electroniques Optiques
M. COMYN Gérard	Informatique Théorique
M. COQUERY Jean-Marie	Psychophysiologie
M. CORIAT Benjamin	Sciences Economiques et Sociales
Mme CORSIN Paule	Paléontologie
M. CORTOIS Jean	Physique Nucléaire et Corpusculaire
M. COUTURIER Daniel	Chimie Organique
M. CRAMPON Norbert	Tectolique Géodynamique
M. CROSNIER Yves	Electronique
M. CURGY Jean-Jacques	Biologie
Melle DACHARRY Monique	Géographie
M. DEBRABANT Pierre	Géologie Appliquée
M. DEGAUQUE Pierre	Electronique
M. DEJAEGER Roger	Electrochimie et Cinétique
M. DELAHAYE Jean-Paul	Informatique
M. DELORME Pierre	Physioliogie Animale
M. DELORME Robert	Sciences Economiques
M. DEMUNTER Paul	Sociologie
M. DENEL Jacques	Informatique
M. DE PARIS Jean Claude	Analyse
M. DEPREZ Gilbert	Physique du Solide - Cristallographie
M. DERIEUX Jean-Claude	Microbiologie
Melle DESSAUX Odile	Spectroscopie de la réactivité Chimique
M. DEVRAINNE Pierre	Chimie Minérale
Mme DHAINAUT Nicole	Biologie Animale
M. DHAMELINCOURT Paul	Chimie Physique
M. DORMARD Serge	Sciences Economiques
M. DUBOIS Henri	Spectroscopie Hertzienne
M. DUBRULLE Alain	Spectroscopie Hertzienne
M. DUBUS Jean-Paul	Spectrométrie des Solides
M. DUPONT Christophe	Vie de la firme (I.A.E.)
Mme EVRARD Micheline	Génie des procédés et réactions chimiques
M. FAKIR Sabah	Algèbre
M. FAUQUAMBERGUE Renaud	Composants électroniques

M. FONTAINE Hubert	Dynamique des cristaux
M. FOUQUART Yves	Optique atmosphérique
M. FOURNET Bernard	Biochimie Sturcturale
M. GAMBLIN André	Géographie urbaine, industrielle et démog.
M. GLORIEUX Pierre	Physique moléculaire et rayonnements Atmos.
M. GOBLOT Rémi	Algèbre
M. GOSSELIN Gabriel	Sociologie
M. GOUDMAND Pierre	Chimie Physique
M. GOURIEROUX Christian	Probabilités et Statistiques
M. GREGORY Pierre	I.A.E.
M. GREMY Jean-Paul	Sociologie
M. GREVET Patrice	Sciences Economiques
M. GRIMBLOT Jean	Chimie Organique
M. GUILBAULT Pierre	Physiologie animale
M. HENRY Jean-Pierre	Génie Mécanique
M. HERMAN Maurice	Physique spatiale
M. HOUDART René	Physique atomique
M. JACOB Gérard	Informatique
M. JACOB Pierre	Probabilités et Statistiques
M. Jean Raymond	Biologie des populations végétales
M. JOFFRE Patrick	Vie de la firme (I.A.E.)
M. JOURNEL Gérard	Spectroscopie hertzienne
M. KREMBEL Jean	Biochimie
M. LANGRAND Claude	Probabilités et statistiques
M. LATTEUX Michel	Informatique
Mme LECLERCQ Ginette	Catalyse
M. LEFEBVRE Jacques	Physique
M. LEFEBVRE Christian	Pétrologie
Melle LEGRAND Denise	Algèbre
Melle LEGRAND Solange	Algèbre
M. LEGRAND Pierre	Chimie
Mme LEHMANN Josiane	Analyse
M. LEMAIRE Jean	Spectroscopie hertzienne
M. LE MAROIS Henri	Vie de la firme (I.A.E.)
M. LEROY Yves	Composants électroniques
M. LESENNE Jacques	Systèmes électroniques
M. LHENAFF René	Géographie
M. LOCQUENEUX Robert	Physique théorique
M. LOSFELD Joseph	Informatique
M. LOUAGE Francis	Electronique
M. MAHIEU Jean-Marie	Optique-Physique atomique
M. MAIZIERES Christian	Automatique
M. MAURISSON Patrick	Sciences Economiques et Sociales
M. MESMACQUE Gérard	Génie Mécanique
M. MESSELYN Jean	Physique atomique et moléculaire
M. MONTEL Marc	Physique du solide
M. MORCELLET Michel	Chimie Organique
M. MORTREUX André	Chimie Organique
Mme MOUNIER Yvonne	Physiologie des structures contractiles
Mme MOUYART-TASSIN Annie Françoise	Informatique
M. NICOLE Jacques	Spectrochimie
M. NOTELET Françis	Systèmes électroniques
M. PARSY Fernand	Mécanique

M. PECQUE Marcel	Chimie organique
M. PERROT Pierre	Chimie appliquée
M. STEEN Jean-Pierre	Informatique

A mon père et ma mère,
leur petit dernier...

Je remercie le Professeur Jean Françon d'avoir accepté de présider le jury et, malgré la rapidité de nos échanges, de m'avoir confirmé dans l'intérêt de notre travail sur la rotation approchée.

Je remercie madame Irène Zambettakis d'avoir rapporté cette thèse. Je tiens ici à lui exprimer toute ma reconnaissance tant pour ses critiques constructives que pour l'intérêt témoigné à nos travaux, mais aussi pour son érudition rare dans ce domaine.

Je remercie le Professeur Max Dauchet. Le verbe à lui seul en dit assez sur le profond respect que je cultive pour ses idées géniales et inspirées, qui ont fait de ce projet une réalité. Qu'il sache combien il me fut agréable et formateur, au sens noble de l'éducation, de travailler avec lui. Au delà des idées partagées, il me reste l'image d'un grand chercheur, passionné et passionnant.

Je remercie le Professeur Michel Mériaux. Alliant le sérieux à l'humour et l'enthousiasme, il est l'autre protagoniste de ce projet. Il en a soutenu tous les développements et contribua de manière importante à la direction de mes travaux. Je lui témoigne ici ma plus profonde reconnaissance pour sa confiance et son aide éclairée. Qu'il perçoive à travers mes mots, mon admiration pour ses travaux et son esprit d'ouverture.

Je remercie monsieur Alain Salmon, responsable du département des développements logiciels de la société Matra MS2I, d'avoir accepté d'examiner cette thèse et d'y avoir apporté le regard original de l'industriel.

Je remercie Michel Binse pour le travail d'équipe partagé au gré de nos réunions hebdomadaires. Je l'encourage pour la rédaction de sa thèse, et lui confie mon admiration pour ses multiples activités. Puisqu'ils firent partie à diverses occasions du projet, je remercie ici Denis Bouhineau, élève de l'Ecole Normale Supérieure de Lyon, brillant intervenant pour les problèmes de complexité; Franck Lelong et Jean-Jacques Vandewalle, étudiants doués qui donnèrent un "habit" neuf à PASTIS (selon leurs propres dires) et mirent au point un algorithme difficile de reconstruction; Patrick Musquer et René Pathenay, élèves-ingénieurs de l'EUDIL, qui réalisèrent le deuxième circuit ASIC.

Je remercie l'équipe graphique pour son enthousiasme et son amitié; qu'elle apprenne ici combien j'admire leurs travaux et leur souhaite rapidement une plus large reconnaissance. Je tiens tout spécialement à remercier Samuel Degrande, sans qui mes débuts en CAO se seraient certainement révélés plus difficiles.
Je remercie l'ensemble des enseignants et chercheurs du laboratoire pour leur accueil sympathique et leurs compétences variées et indiscutables. Que Jean-Luc, Hafid et Stéphane soient remerciés de m'avoir offert leur amitié,... ou une place dans leur bureau. Je remercie les membres de l'équipe technique, dont plusieurs savent déjà combien j'estime grandement leur travail et leur efficacité. Je remercie enfin le personnel du laboratoire pour l'aide quotidienne qu'il nous apporte.
A côté de la recherche, ces années furent pour moi l'occasion de me "frotter" à l'enseignement et aux étudiants. Je remercie donc les enseignants de l'ISEN pour leur confiance. Fasse que le spectre des copies à corriger disparaisse irrémédiablement...

Je remercie enfin parents, frères et soeurs, amis qui m'ont aidé au long de ces trois années.
La recherche est faite d'optimisme et d'enthousiasme, Christelle en a partagé aussi des moments plus pénibles de vide et d'incertitude. Qu'elle reçoive ici le témoignage de tout mon amour pour ce partage inconscient.

Introduction

chapitre 1

Architecture de Machines de Vision

chapitre 2

L'extraction de segments

chapitre 3

Les Modules Architecturaux Dédiés

Conclusion

En Annexe: la simulation en langage C du processeur de rotation rapide

Nota: les références bibliographiques sont données à la fin de chaque chapitre; celles-ci sont en effet relativement indépendantes d'un chapitre à l'autre.

Introduction

Les images, consommées à l'excès ou reconstruites à l'occasion de nos lectures, façonnent inconsciemment notre perception du monde. Déposées sur le papier, elles reflètent nos appréhensions et compréhensions des objets et des êtres qui nous entourent. L'importance de celles-ci rejoint celles des mots qui les repèrent, les cernent ou les figent. Chaque mot, chaque image, mais aussi chaque son et chaque odeur constituent ce mélange doux, violent, rarement neutre, que nous construisons intérieurement à chaque découverte. La vision est alors ce lien privilégié qui, du monde extérieur, nous projette une image intérieure, sensible et conçue.

chapitre 1: Architecture de machines de vision

Loin de cette réalité intangible, les chercheurs tentent de comprendre le fonctionnement de cette machine extraordinaire, qui permet à l'homme de reconnaître en une dizaine d'étapes les objets de son univers. Les enjeux, qui s'attachent à ces recherches, sont impressionnants: ils donneraient à nos machines aveugles et esclaves la même "vision du monde" que la nôtre. Des véhicules autonomes à la poursuite de cibles, mais aussi de la reconstruction d'images médicales au contrôle de fabrication, les applications ne manquent pas. Cependant, avec sa centaine de millions de photorécepteurs par rétine, et une reconnaissance d'objets en une soixantaine de milli-secondes, le cortex visuel dépasse de loin les possibilités techniques actuelles.

Le premier chapitre de cette thèse est l'occasion d'analyser un peu plus finement le processus de vision. De cette analyse émerge la nécessité d'une architecture spécialisée, qui à la manière de notre système visuel s'appuierait sur une succession d'étapes de traitement (les informaticiens parlent de "pipeline"), et un très haut niveau de parallélisme pour chacune des étapes. Ce chapitre se propose donc de redécouvrir les machines qui ont révolutionné ou simplement marqué l'évolution matérielle des outils de vision.

chapitre 2: L'extraction de segments

Lors de cette thèse, nous nous sommes appliqué à mettre en oeuvre un paradigme simple: réduire la complexité des méthodes en approchant celles-ci. Dès le début, nous avions identifié la transformation dont la complexité semblait la plus élevée: la rotation d'images. Nous avons donc cherché un nouvel algorithme de rotation, qui verrait sa complexité réduite par approximation de la transformation. Appliquée ensuite à l'extraction de segments et à la vectorisation, c'est à dire à l'obtention d'une description sous forme de droites de l'image traitée, nous avons proposé un algorithme original dont nous montrons qu'il se rapproche à la fois de la transformée de Hough et de la dilatation morphologique. Le chapitre 2 présente donc les différentes techniques d'extraction de segments et de vectorisation existantes, et se clôt sur notre proposition cellulaire utilisant l'*à-peu-près*.

chapitre 3: Les modules architecturaux dédiés

Le dernier chapitre est l'aboutissement du paradigme précité, et propose une architecture dédiée pour la rotation rapide. A cette description architecturale, concrétisée par la réalisation de deux circuits ASICs, s'ajoutent des considérations sur l'architecture générale de la machine PASTIS, mais aussi plus particulièrement sur sa mémoire d'images, qui supporte une partie non négligeable de la complexité de l'extraction de segments.

Chaque fonction retenue lors des évaluations logicielles précédentes, trouve une réalisation matérielle grâce au concept MAD: Module Architectural Dédié. Quoique nous ne proposions qu'un seul MAD complètement conçu, celui de la rotation rapide, le lecteur découvrira dans ce chapitre 3 la matière à concevoir ou créer de nouveaux MADs. Notre démarche particulière trouve son dénouement cohérent dans ces réflexions ou descriptions, qui rebuteront peut-être certains par leur aspect technique.

Bien que chacun des trois chapitres s'articule indépendamment des deux autres, seule la lecture complète permettra de s'imprégner des contraintes importantes et variables des traitements d'images, mais aussi de percevoir les fondements de nos travaux: parallélisme et approximation. L'architecture et l'algorithmique sont deux côtés de la nature informatique que nous avons voulu fondre dans notre étude. Que le lecteur retrouve chacun de ces aspects est notre seul voeu.

"L'étrange facilité qu'ont les objets créés par l'homme pour finir par se ressembler ne s'est pas encore révélée à eux"

M.Yourcenar

chapitre 1

Architecture de Machines de Vision

Nous présentons un amalgame des différentes machines utilisées en vision, et conçues pour la vision depuis que l'informatique crée des machines dédiées. De ce dédale initiatique, on retiendra surtout les différences entre les diverses architectures connues de nos jours, et leur utilité et aptitude à résoudre certains problèmes. Partant du principe que traiter de l'information, c'est avant tout se donner une suite d'actions qui, appliquées à un groupe d'éléments, le déforment ou en créent un nouveau, cette seconde partie démontre que pour la vision, et à un niveau donné dans ses phases de traitement, le support physique qui déroule ces actions induit des contraintes importantes sur les temps de réponse espérés.

Ainsi nous voyons se renforcer l'idée du choix de l'architecture en fonction des traitements effectués. Les modèles les plus répandus sont le modèle SIMD avec l'existence de machines novatrices depuis ces vingt dernières années, mais aussi le modèle systolique qui a pleinement profité, et profitera encore, de l'arrivée des techniques VLSI et ULSI, notamment avec le développement des ASICs (*Application-Specific Integrated Circuits*). Enfin, au niveau symbolique, le recours au modèle MIMD ou à ses variantes constitue un moyen efficace de diminuer la complexité des algorithmes et heuristiques mis en oeuvre pour décrire et analyser la scène.

chapitre 1

Architecture de Machines de Vision

1 Introduction

Dans le monde informatique, l'architecture des machines répond tout uniment aux multiples besoins de rapidité, d'efficacité et de puissance. Toutefois l'existence de processeurs dits d'usage général oblige les *céléropathes* à préciser les ordres de grandeur de leurs traitements dans un jargon trop sophistiqué: mips, gflops, nano ou pico-secondes,... et surtout à vouloir toujours un peu plus que ce que la technologie leur offre *hic et nunc*. D'aucuns peuvent vous parler de parties d'échec millénaires, ce que tout monopolyste convaincu appréhendera sans difficulté...

Aussi pour obvier à une critique simpliste mais répandue: "est-ce pour vous amuser que vous faites de l'architecture?", il nous a semblé utile de rassembler les idées qui nous amènent à ce métier d'architecte. Cette réflexion est d'autant plus encourageante qu'elle permet le survol des concepts des machines créées, prémices insoupçonnables des évolutions et révolutions à venir.

1.1 Nécessité d'une architecture spécialisée

Les performances croissantes des technologies d'intégration amènent le doute dans la communauté informatique. L'architecture est souvent renvoyée à son acception restreinte de la mise en oeuvre d'un ou plusieurs processeurs d'usage général, qui pense-t-on suffisent amplement à nos besoins. Cette réalité s'est diffusée rapidement en informatique graphique, lorsque fut franchi le seuil symbolique et inconsistant des 25 MHz; comprenez vingt-cinq millions de pixels par seconde. Dès lors, beaucoup s'étonnent de ce qu'apparaissent ici et là des projets de processeurs dédiés, à une époque où la propension au parallélisme massif à base de processeurs d'usage général s'érige en modèle manichéen du progrès.

L'objectif de cette courte synthèse sur les machines de vision est de démontrer au lecteur que les systèmes de vision évoluent insensibles aux progrès techniques, et que le parallélisme reste une obligation, sinon une contrainte. Ce sentiment d'immobilisme trouve sa source dans l'approche théorique que l'on a de la vision -les hommes commencent à peine à comprendre ce qui fait que nous voyons-, mais aussi dans une certitude technique: l'homme possède une machine de vision peu rapide mais terriblement efficace, grâce notamment à son parallélisme intrinsèque.

Pour débuter cette présentation, nous avons imaginé un traitement qui pourrait être à la base du processus de la reconnaissance de scènes, et analysé les contraintes sur chacune des tâches de ce processus. Toutefois, nous précisons ne pas savoir si ce processus existe et donne le résultat escompté. Ce que cet exemple doit nous révéler, procède de l'idée suivante: chacun des modèles parallèles de calcul, auxquels on associe bien sûr une architecture spécifique, intervient dans ce processus global. Faut-il alors en déduire qu'une bonne machine de vision possède une architecture hétéroclite?

1.1.1 Processus de vision

Nous analysons les étapes successives d'un processus de reconnaissance de formes, à partir d'une image et à l'aide d'une base de données d'objets reconnus. Dans le cas le plus général, les étapes sont alors les suivantes:

(i) *étape de traitement de l'image*, dite aussi de bas-niveau: l'image est filtrée de son bruit, et traitée en vue de renforcer des caractéristiques utilisées par la suite;

(ii) *étape d'extraction d'information*: l'image est analysée sur la base des traits caractéristiques précédemment dégagés; l'espace image se transforme en un espace information (soit vectoriel, soit morphologique,...);

(iii) *étape de reconnaissance:* le contenu de l'image décrit dans l'espace information est analysé sur la base de ses caractéristiques, pour rechercher la signature de formes connues par le système de vision, et susceptibles d'être présentes dans l'image.

Ce processus est cohérent avec les descriptions qu'en proposent les spécialistes du domaine. Il n'est qu'à ouvrir le célèbre [BALL82] et feuilleter les quelques cinq cent pages de cet ouvrage, pour retrouver dans le sommaire même une découpe semblable d'un (ou du) processus global de vision. L'étape d'extraction d'information correspond dans ce livre à celle d'analyse des structures géométriques, tandis que celle de reconnaissance correspond à l'analyse des relations, et de leurs structures.

1.1.2 Etape de traitement de l'image

Cette étape est difficile à partager de la suivante; selon nous, la question posée qui les distingue entre elles, est la suivante: "y-a-t'il ou non un changement d'espace de représentation?". En effet dans cette étape, la seule information qui évolue est le contenu de l'image. Par contre, lors de la phase d'extraction d'information, on passe de l'espace de représentation *pixel* à un nouvel espace: celui de la description paramétrique des formes à partir des primitives du système de vision. Bien que la frontière entre ces deux étapes soit réellement mal précisée, puisque tout traitement n'est jamais innocent, elles ne peuvent être pensées de la même manière, comme nous l'allons voir.

Cette première étape se caractérise par la quantité importante de pixels à traiter. Même s'il s'agit d'une information pauvre[1], le nombre de pixels traités par image est de l'ordre du million (250K pour une image 512x512, 1M pour 1024x1024). Aussi on peut écarter d'emblée toute méthode séquentielle, puisqu'à l'heure actuelle les unités fonctionnelles de calcul ont des temps de cycle de l'ordre de 10, 20 ou 30 nano-secondes, ce qui signifierait un temps de calcul pour toute l'image de 10, 20 ou 30 milli-secondes... en considérant qu'une seule opération suffise pour chaque pixel. L'inconsistance du seuil cité précédemment des 25 MHz, ou 40 nano-secondes, provient justement de cet axiome irréaliste qui voudrait qu'on n'effectue qu'une seule opération par pixel traité.

1. Il est admis qu'un nombre réduit de niveaux de gris suffit à traiter correctement les images. Ce nombre est souvent de 256, c'est à dire 8 bits par pixel, mais peut descendre à 16 pour des traitements automatiques, donc 4 bits par pixel.

Tenir un temps de cycle de 20ns est d'autant plus improbable qu'il faut généralement accéder au pixel par le biais d'une lecture puis d'une écriture mémoire. Cette perte de temps due à la mémoire -sans aborder les conflits inévitables d'une approche parallèle-, a conduit une équipe de notre laboratoire à concevoir une machine de synthèse d'images qui n'utilise justement pas de mémoire avant affichage: il s'agit du projet IMOGENE. Par symétrie, nous pourrions envisager de ne pas en utiliser entre la saisie (caméra) et les processeurs de traitement. Le cytocomputer [STER81] est une machine pipeline qui traite les pixels sur une chaîne de 80 processeurs logiques. Ces traitements complètement recouverts éviteraient d'avoir recours à une mémoire d'images, et de traiter celles-ci au rythme vidéo [ZAVI86]. Toutefois, il n'est pas démontré que le processus de vision soit non bouclé. Et, si le pipeline s'avère une bonne base architecturale, il faudra peut-être envisager d'y insérer de la mémoire.

On le voit, pour un 'bête' programme d'une dizaine d'instructions par pixel, le traitement de l'image se fera dans un laps de temps d'une seconde environ, limité surtout par les accès mémoire. Pour chaque tâche effectuée, ce nombre important d'opérations arithmétiques, agencées simplement, aboutit à des systèmes de vision basés sur des unités fonctionnelles puissantes où les notions de micro-programmation, de mémoire cache et de processeurs câblés cohabitent. La vision occupe alors une partie du domaine du calcul intense, surtout pour les tâches de niveau bas. Implantés sur des machines SIMD pipelines, par exemple CRAY2, ou tableaux, MPP, les algorithmes demandent des architectures sophistiquées, et donc chères, si l'on souhaite obtenir des temps de réponse convenables.

Toutefois, nombre d'applications de traitement d'images ont un fonctionnement qualifié de temps réel, sur des processeurs accessibles (où le temps réel fait ici référence à la vision par l'homme: $1/25^{ème}$ de seconde par image). Ce constat se comprend grâce aux propriétés des processus mis en oeuvre. Deux caractéristiques sont en effet essentielles à ce sujet: l'indépendance et la localité. La localité des tâches élémentaires, celles effectuées pour chaque pixel, offre à l'architecte une organisation simple des communications entre pixels lors des calculs. L'indépendances de ces tâches élémentaires permet d'exécuter celles-ci en parallèle, sans interférence.

Utiliser à bon escient ces deux propriétés est l'objectif des machines SIMD tableaux, mais aussi des machines systoliques. La difficulté est de situer l'algorithme dans ceux que [KUNG82] différencie: les algorithmes limités par le calcul (*compute-bound*) et ceux limités par les entrées-sorties (*I/O-bound*). Une approche systolique permet de multiplier les unités fonctionnelles dans le premier cas. Les techniques d'entrelacement des bancs mémoire offrent une réponse aux problèmes d'entrées-sorties.

Cependant les algorithmes doivent se prêter à ces modélisations: chaque problème présentant ses propres caractéristiques. Les algorithmes qui se prêtent le mieux à une modélisation du type SIMD sont: les techniques de filtrage et convolutions, l'extraction des contours par gradient, les méthodes d'analyse morphologique des images: dilatation, érosion, filtrage, squelette,...

Enfin, l'étape de traitement de l'image est celle qui a évolué le plus rapidement grâce au développement du VLSI. Son intérêt est vital pour l'utilisation d'images réelles enregistrées par un capteur. L'objectif des traitements est de corriger les défauts dus à un mauvais éclairage, à une mauvaise résolution du capteur, à un flou sur l'image (image ou partie d'image bougée), à une mauvaise mise au point,... Il existe de bons produits grand public de traitement d'images: avec le captage (ou la capture), le filtrage et des traitements élémentaires sur image, généralement construits autour d'un bus polyvalent, VME par exemple, qui relie les cartes à une carte CPU.

1.1.3 Etapes d'analyse, d'extraction de primitives

Cette étape est décisive dans le modèle de vision hiérarchisé. En effet, non utilisée dans l'approche associative, cette étape devient nécessaire dans une approche structurelle, où chaque figure est décrite par un ensemble de primitives et de relations entre elles. Cette étape extrait donc ces primitives dans l'image, et les relie entre elles sur des critères morphologiques: combinaison de primitives présentes mais n'appartenant pas à la même classe, ou topologiques: position, taille,... des primitives.

Cette étape se caractérise par un nombre important de pixels à traiter, par un changement de l'espace de représentation: de l'image pixel à l'image "primitive-relation", et par des algorithmes essentiellement séquentiels: le suivi de contour est un bon exemple, l'approximation polygonale. L'algorithme doit ici diminuer fortement et *intelligemment* la quantité d'information: recherche de la meilleure description avec la plus faible quantité de descripteurs.

Toute la difficulté consiste à trouver un aspect parallèle à ces algorithmes, ou d'en trouver de nouveaux fonctionnellement proches de ceux utilisés fréquemment. Deux points sont sensibles: l'espace de représentation finale correspond le plus souvent à un graphe, et l'algorithme tient plus de l'heuristique. Concernant le graphe, il est évident que celui-ci reste d'une mise en oeuvre difficile sur les machines actuelles. De plus, la construction du graphe primitive-relation absorbe une partie importante de la complexité du traitement global, ce qui amène l'introduction d'heuristiques de construction. Ces heuristiques fonctionnent sur des modèles difficiles à borner en temps: essai-erreur, fusion-éclatement, prédiction-vérification,...

De ces deux contraintes, les développeurs tentent de conjuguer harmonieusement leurs incidences en algorithmique et architecture. Le problème actuel reste d'ordre algorithmique: ainsi nous verrons que les études menées sur l'approximation polygonale sont en fait des réalisations logicielles de celle-ci. Contrairement à l'étape précédente, peu d'algorithmes sont gourmands en calcul; mais tous le sont en accès mémoire générés: la construction finale y est pour beaucoup. Pour revenir à l'absorption d'une partie de la complexité par cette construction, elle est triviale et signifie simplement que plus le modèle est avancé, souple et donc facile à exploiter, plus il est difficile à construire.

Notre travail se situe donc à ce niveau, qui nous a offert un vaste terrain d'expérimentation de solutions originales, susceptibles de diminuer la complexité des actions. Une fois posés les objectifs recherchés, et puisque sont presque connus les moyens de les atteindre, le progrès consiste à découvrir quels sont les degrés de liberté du concepteur. Et à mettre ceux-ci à profit...

1.1.4 Etapes de reconnaissance

Lors de la phase de reconnaissance, la quantité d'information a fortement diminué, fruit de l'étape précédente. Toutefois, le développeur bute ici sur la complexité des algorithmes, étroitement liés à ceux d'isomorphismes de graphes. La relaxation, le recuit simulé, la programmation dynamique pour le parcours d'arbre,... sont des méthodes utilisées dont on ne peut retenir que leur complexité en l'état actuel de nos connaissances. Pour ne prendre qu'un exemple, si le recuit simulé offre de bonnes solutions en physique des cristaux, le parallélisme offert et l'efficacité de l'algorithme dans un monde non convexe, sont limités par la projection du problème sur une machine parallèle, dont on prend conscience après coup soit de la monstruosité (le cas du problème du voyageur de commerce pour N~100 est significatif) soit de son inaptitude (le temps est ici la limite à laquelle nous pensons).

Actuellement la plupart des algorithmes autorisent un parallélisme asynchrone, dû au caractère local de certains traitements sur des sous-graphes. Ce parallélisme est alors trop dépendant de la construction du graphe traité, pour qu'il s'étende à tout type de formes. Les graphes planaires renforcent cependant cette localité des traitements, tout en limitant l'explosion des choix par sa rigidité. L'étude reste ouverte.

1.2 Survol des architectures pour la vision

Passée cette présentation des traitements, nous analysons maintenant le sens et la réalité du terme "parallélisme" dans le domaine de la vision. Sans entrer dans les détails, ni faire un $n^{ième}$ état de l'art sur les machines parallèles, nous avons cherché ce qui existait dans ce domaine et pour la recherche. Avant de parler plus en détail des concepts sous-jacents aux machines parallèles, [PATT90, p.588] différencie en terme productique une évolution d'une révolution en informatique:

> *"Revolutionary ideas are easier to publish than evolutionary ideas, but to be adopted they must have a much higher payoff. Caches are an example of an evolutionary improvement. Within five years after the first publication about caches almost every computer company was designing a machine with a cache. The RISC ideas were nearer to the middle of the spectrum, for it took closer to ten years for most companies to have a RISC product. An example of a revolutionary computer architecture is the Connection Machine. Every programs that runs efficiently on that machine was either substantially modified or written especially for it, and programmers need to learn a new style of programming for it. Thinking Machines was founded in 1983, but only a few companies offer that style of machine."*

Cette différence exprime précisément, selon nous, l'intérêt que présente les développements de machines parallèles en vision. Le parallélisme participant d'une nécessité incontournable, ces machines doivent être analysées sur deux plans, théorique et technique. Certaines ne sont qu'une évolution qu'offre la technique actuelle (machines MIMD orientées temps partagé), d'autres apparaissent comme de petites révolutions reprises et affinées (machines SIMD, systoliques).

1.2.1 Analyse du parallélisme

[DANI81] expose sa propre analyse de la dimension du parallélisme en traitement d'images. Cette dimension fait référence à une analyse descendante d'une tâche non explicitée. Développée sur une machine séquentielle 1 bit, cette tâche comporte selon lui quatre boucles DO imbriquées, pour:

(i) la séquence des opérations élémentaires ou instructions à effectuer,

(ii) les coordonnées des points dans l'image de sortie pour chacune des opérations,

(iii) les pixels 'voisins' dans l'image d'entrée pour chacun des pixels de sortie,

(iv) les bits individuels de chacun des pixels du voisinage sur notre machine 1 bit.

En notant k_i (resp. k_{ii}, k_{iii}, k_{iv}) le nombre d'actions de niveau (i) exécutées en parallèle, on obtient facilement une mesure du parallélisme global qui est la suivante: $K = k_i \times k_{ii} \times k_{iii} \times k_{iv}$, qui évalue grossièrement celui d'une machine de traitement d'images. L'excursion des paramètres n'est pas la même, en voici une idée:

nombre d'opérations par tâche:	*1*	*100*
taille de l'image:	*64 x 64*	*10000 x 10000*
points du voisinage:	*1*	*200*
bits par pixel traité:	*1*	*64*

Plus intéressante comme réflexion est celle qui consiste à se demander quelle architecture et quels problèmes cache chacun de ces paramètres. Cette synthèse, sous forme de tableaux, guidera les pages à venir. Toutefois, notons que le type d'opérations envisagées est ici implicite: toutes les opérations sur l'image. La catégorie de traitements est donc la première (cf. 1.1.2), ou la seconde (cf. 1.1.3) pour certains d'entre eux. Voyons maintenant ce tableau:

Parallélisme des opérations	Pipeline	machine Pipeline: Cytocomputer machine Systolique: iWarp machine Pyramidale: Sphynx ... bus en anneau: Capitan
Parallélisme sur l'image	SIMD	machine tableau: Illiac IV, MPP tableau+hypercube: Connection Machine
Parallélisme sur le voisinage	Commun.	LOCALE: machines tableaux, ou cellulaires GLOBALE: hypercube (PASM)
Parallélisme sur le ratio Pixel/Bit	Coût	ASICs tableaux: GAPP, ELSA machines vectorielles: Illiac IV, Cray

Si le *ratio* Pixel/Bit, qui indique le nombre de bits que peut traiter simultanément une machine, se situe aux confins des préoccupations scientifiques, par contre il n'en est pas de même des contraintes induites par les trois autres formes de parallélisme. Le parallélisme sur l'image s'est approprié les découvertes sur les machines SIMD, si bien que dorénavant on parle de processeurs tableaux (GAPP, ELSA), celles-ci ayant dépassé le seuil critique de l'intégration. Les propositions de tels processeurs indiquent combien ce concept est parfaitement adapté. Plus difficile, le parallélisme sur le voisinage introduit la notion de communication qui reste éminemment complexe dans les structures actuelles.

Deux philosophies se partagent les réalisations viables: les machines à communication locale, généralement limitée aux quatre, six ou huit premiers voisins, ou fixée entre cellules dans le modèle cellulaire; et les machines à communication globale, groupe qui va des bus en anneau, arbres, réseaux à entrelacement (Oméga, hypercube,...), réseaux banians, matrices de commutation (crossbar, réseau de clos). Quant au dernier parallélisme, celui sur les opérations, il apparaît de plus en plus traité et développé, du fait notamment des performances actuelles d'intégration qui permettent de réaliser facilement des processeurs intégrant les deux parallélismes précédents.

1.2.2 Le modèle SIMD

Ce modèle se prête aux traitements pixel, ou traitements par plan de bits simultanés. Il se caractérise par un seul flux d'instructions exécutées sur plusieurs flux de données. Pour des programmes exécutant les mêmes opérations sur un ensemble important de données, la machine SIMD présente un avantage sérieux par rapport à la simple multiplication des unités d'exécution.

En effet, une unité d'exécution se compose essentiellement d'une unité de contrôle, d'une unité fonctionnelle arithmétique et logique, voire plusieurs, et d'un ensemble de registres, de travail et d'état de l'unité fonctionnelle, que nous pouvons considérer comme une mémoire de données. Nous n'avons pas voulu citer la mémoire de programme, attendu que la machine dont nous parlons ne 'crée' pas de codes (on utilise une ordinateur hôte pour charger le code dans notre machine, ce qui en vision est plausible compte tenu de la spécialisation évidente des machines). Aussi, si nous considérons que ce code se trouve dans une mémoire appartenant à l'unité de contrôle, la machine SIMD évite l'implantation de N unités de contrôle, chacune possédant une petite mémoire de code.

Dans l'état actuel de la technologie, où nous le rappelons la mémoire occupe un espace physique qui se restreint certes, mais reste non négligeable (et coûteux) dès que cet espace mémoire est de taille importante, l'architecture SIMD réalise non seulement un gain au niveau du séquenceur des instructions mais bien plus au niveau de la mémoire de code. Ce code n'est pas dupliqué N fois.

Cependant, et avant d'en terminer avec ce partage du code, l'architecture SIMD trouve là sa pierre d'achoppement: le branchement conditionnel. En effet, dès lors que chaque unité fonctionnelle travaille sur ses propres données, elle peut se retrouver dans une situation particulière qu'un modèle de contrôle global ne peut plus appréhender. La réponse vient de la possibilité offerte aux unités fonctionnelles de ne pas exécuter certaines instructions du programme. Cette technique demande une duplication dans le temps du code exécuté, pour les deux cas de la condition vraie et de la condition fausse.

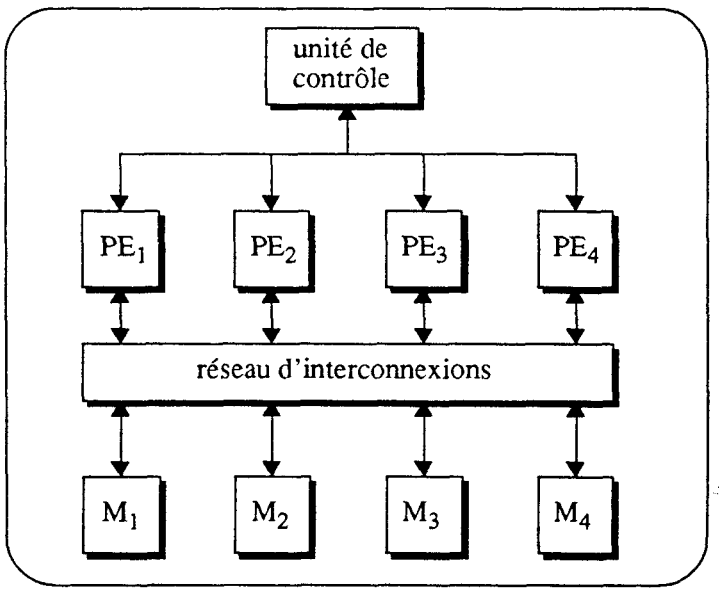


figure 1 le modèle SIMD

La machine SIMD est donc composée de processeurs élémentaires, PE, qui regroupe l'unité fonctionnelle ainsi que l'ensemble des données (mémoire locale de travail). Le fonctionnement synchrone des PEs est géré par une unité de contrôle, unique. L'architecture SIMD repose beaucoup sur le choix du modèle de communication entre PEs, qui contraint la gestion mémoire de la machine.

Pour conclure, la capacité des PEs et leur nombre est un choix difficile. A une époque où les capacités d'intégration offertes par la technologie VLSI CMOS sont importantes, cette question est encore plus cruciale: est-il plus intéressant d'intégrer un nombre croissant (et grand) de processeurs 1 bit (processeurs GAPP et ELSA) ou un nombre plus restreint de processeurs plus puissants (32 bits, à la manière de PASM). Les nouveaux microprocesseurs des deux *leaders* du marché: *Intel* et *Motorola*, intègrent quelque 1,2 millions de transistors, et le noyau CPU est de plus en plus petit -actuellement 30% en surface- du fait du recours à la technique RISC en interne. Aussi une unité fonctionnelle et sa mémoire de travail ne prennent désormais qu'une place infime, si on travaille sur 1 bit.

1.2.3 Le modèle MIMD

Cette architecture appartient à une partie refoulée de notre mémoire collective; 'notre' se réfère à la communauté informatique. En effet, dès les débuts de l'informatique, beaucoup pensèrent que la mise en parallèle d'un grand nombre de processeurs serait un bon moyen d'accroître fortement la puissance de calcul des machines. Hélas, si *un plus un* égale en toute bonne logique *deux*, rapidement les architectes se sont aperçus de la *non*-linéarité des *ratio* d'efficacité des machines de calcul. Ainsi un million de processeurs n'offre pas à coup sûr une puissance de calcul d'un million de fois celle du processeur de base, pour une architecture coûtant un peu plus cher...

Plusieurs raisons peuvent être invoquées (voir [PATT90] pour plus de détails). La première tient aux algorithmes développés, dont l'efficacité ne croît pas linéairement avec la taille de la machine utilisée. Ainsi la communication entre processeurs, qui revient finalement à un partage d'objets, reste le point sensible de ces architectures multi-processeurs. Deux solutions existent alors à cette communication: soit la mémoire partagée soit le réseaux de communications inter-PEs. Bien qu'incontestablement la plus appréciable pour ces possibilités d'extension de la machine, et la plus utilisée à l'heure actuelle (CRAY X et Y-MP, IBM 3090VF, Sequent), la mémoire partagée possède son point faible incontournable: la différence de temps de fonctionnement entre celle-ci et les CPUs. Ce problème apparaît d'autant plus insurmontable qu'intrinsèquement les modèles de mémoire actuels ralentissent déjà les processeurs. Comment peut-on dès lors imaginer un modèle d'architecture MIMD qui permette de tirer partie au mieux des N processeurs travaillant en parallèle? Les techniques d'entrelacement de la mémoire, constituée de bancs séparés, connaîtront à un moment donné leur limite.

De plus elles fourniront insidieusement une autre raison à l'inefficacité des machines parallèles. En effet, si la réponse apportée consiste à optimiser ces communications inter-processeurs, en ayant un *a priori* important quant à l'adéquation des solutions matérielles de celles-ci, lors de la phase de programmation, la restriction induite déplacera le problème vers une meilleure qualification du programmeur. Celui-ci devra posséder pleinement les aspects architecturaux, combien sophistiqués, de sa machine. On sait d'ores et déjà combien cette contrainte gêne la diffusion 'grand public' de telles machines, puisque les machines vectorielles utilisent ce type de programmation d'initiés: au niveau du pipeline vectoriel d'une CPU. La réalisation d'un programme parallèle adapté à une machine vectorielle donnée peut nécessiter un temps de développement long; la connaissance du programmeur devient trop complexe pour l'implanter dans un compilateur, tel qu'on les conçoit aujourd'hui.

Ainsi, pour prendre un exemple quotidien, la mise au point de compilateurs pour les nouvelles unités RISC (RS6000, i860) pose des problèmes et nécessite des temps de développement important, du fait du parallélisme fonctionnel bas-niveau de ces processeurs capables d'exécuter deux, voire trois instructions par cycle d'horloge. Notons que beaucoup de *mainframes* existent, dotés de deux, trois,..., six processeurs, mais sans apporter de solutions viables en terme de gain, sur l'aspect parallèle des algorithmes développés. Si le gain n'est pas linéaire, où est-il?

En fin de compte, ce modèle d'architecture qui renvoie l'utilisateur face à des choix complexes: avoir un personnel très qualifié, prendre des risques quant aux performances "relatives" des machines, trouver pour une classe d'algorithmes la meilleure machine,... est bien l'un des défis de notre époque. Les machines MIMD sont des machines au comportement algorithmique sensible, mais qui appliquées à un domaine précis, à une classe définie d'algorithmes, peuvent se révéler de très bonnes machines de part leur puissance et aussi leur souplesse d'utilisation; là est tout l'intérêt pour les machines de vision, et celles de traitement du signal plus généralement. Le modèle SPMD est alors bien utile, puisqu'il rejoint les modèles SIMD et MIMD.

Quant aux machines MIMD à réseau de communication (hypercube d'Intel, les machines à base de transputers), elles offrent des gains intéressants pour ces algorithmes qui ne demandent qu'un nombre restreint de communications. Selon [PATT90], les machines à mémoire partagée sont les vraies machines multiprocesseurs, tandis que celles à communication par message appartiennent à la famille des multi-ordinateurs.

1.2.4 Les modèles spécifiques

Le traitement du signal et de l'image a favorisé l'apparition de modèles spécifiques de traitement. Parmi ceux-ci, nous citerons:

(i) les machines pyramidales, qui marient fort à propos les modèles SIMD, MIMD et le pipeline. Leur défaut majeur se situe dans les goulets d'étranglement que crée cette architecture mal adaptée au monde 2D de la réalisation matérielle des circuits.

(ii) les machines systoliques, qui apparaissent à certains comme le modèle MISD [PATT90, 573]. Le systolique permet de réduire favorablement les problèmes extrêmement gourmands en calculs, en offrant l'aspect pipeline lié à de fortes contraintes sur les débits entrée-sortie. Ce concept dérive lentement des applications dédiées du début vers des processeurs systoliques très puissants.

(iii) les machines connexionnistes, qui permettent aux convaincus (?) de modéliser rapidement leurs réseaux et d'espérer des gains substantiels dans les phases d'apprentissage, toujours très longues. De plus, le fonctionnement original de ces réseaux offre une bonne occasion d'investigation du domaine toujours surprenant des machines cellulaires.

Nous terminerons par les machines d'évaluation, construites autour d'un bus rapide. Elles servent de base matérielle de développements de traitements graphiques pour des usages aussi diversifiés que l'industriel, le militaire, le médical,...

2 Architecture SIMD pour la vision

L'objectif de ce paragraphe est de familiariser le lecteur néophyte aux concepts originels des machines SIMD, et plus particulièrement des machines tableaux. Chacun s'accorde en effet sur le rôle particulier de ces machines dans l'évolution informatique; certains parlent même de révolution [PATT90]. Face aux nouveautés inéluctables, telles que la micro-programmation, le pipeline et le cache, la machine massivement parallèle apparaît dans l'horizon de ce siècle comme une issue certes fragile, mais possible aux machines de type Von Neuman. Puisqu'il pouvait beaucoup, il est d'ailleurs remarquable que ce soit ce même Von Neuman qui proposa la première application d'automates cellulaires; application qui se développerait naturellement sur une machine SIMD. C'est peut-être le paradoxe le plus révélateur de l'ambivalence informatique: comment le créateur d'une architecture tellement naturelle qu'elle reste le paradigme, découvrit-il l'algorithme qui la mettait en défaut?

Cette découverte du "parallélisme massif", nous l'aborderons tout d'abord avec une machine restée célèbre ILLIAC IV. Ensuite nous parlerons de CLIP IV, de MPP, dont nous détaillerons alors deux applications de traitement d'images. Enfin nous présenterons la Connection Machine, une machine dont le parallélisme massif n'est pas à démontrer. Ce survol est d'ordre technique, bien qu'on puisse lui reprocher une certaine brièveté. Nous ne voyons pas là la critique majeure. A notre avis, et pour la faire ressortir d'autant plus, la vraie critique serait bien plus relative aux applications. En effet, nous ne détaillerons que trop rarement les applications de ces machines au traitement d'images.

Cette lacune est volontaire parce que le sujet serait alors trop vaste. De plus, les machines SIMD paraissent si naturellement adaptées aux applications bas-niveau du traitement d'images, qu'il ne nous a pas semblé utile de le préciser davantage. Pour une machine ciblée, le développement d'un algorithme devient naïf, et les raffinements trop complexes pour les gains espérés. D'ailleurs, le lecteur soucieux de ce vide trouvera dans le chapitre suivant les applications, qui nous intéressent, sur ces machines.

Pour notre part, nous avons essayé d'étudier certaines architectures SIMD, en posant les bons problèmes, et en voyant comment ils ont été résolus dans les différentes machines conçues. Les questions qu'on peut se poser portent sur:

(i) l'architecture d'un processeurs élémentaire (fonctions, taille),

(ii) la communication entre processeurs élémentaires,

(iii) la programmation d'une machine SIMD.

Nous espérons y avoir apporté les bonnes réponses, ce dont seul le lecteur est juge.

2.1 ILLIAC IV

Directement issue du dessin de la machine SOLOMON de Stolnick, Illiac IV reste, selon [PATT90], le projet de superordinateur le plus infâme (sic: "perhaps the most infamous supercomputer project"). En effet, bien que son architecture fût si novatrice en bien des points, notamment pour les machines SIMD, cette machine se révéla issue d'une rare aberration pour son utilisation comme superordinateur. Son coût final dépassait de quatre fois les prévisions initiales (8 millions de dollars, tout de même...), alors qu'un quart de la machine fut réellement construit, et que les performances s'avérèrent relativement basses: 15 MFlops au lieu des 1000 MFlops espérés [HORD82]).

2.1.1 Architecture générale

Cette machine [BARN68] se compose de 256 processeurs élémentaires organisés en quatre matrices de 64 processeurs élémentaires (PE) chacune. Chaque processeur accède une mémoire locale de 2K mots[1]. Les caractéristiques de cette machine les plus intéressantes sont:

-un mode d'opérations conditionnelles: un bit Enable contrôle localement l'exécution de l'instruction au niveau du processeur. Ceci permet de remplacer les branchements conditionnels dans cette architecture SIMD. Apparu dans la machine SOLOMON, ce bit de validité du processeur élémentaire a permis la mise en oeuvre du fonctionnement conditionnel, essentiel aux machines de type Von Neuman, et qu'il fallait donc implanter dans les machines SIMD.

-un routage: chaque processeur communique avec 4 de ses voisins (le processeur i communique avec i-1, i+1, i-8, i+8, ses 4 plus proches voisins en maillage carré). La communication se fait par mot complet (64 bits). Les communications entre PEs à distance quelconque l'un de l'autre se font par une séquence de routages du mot contrôlé par une seule instruction. Sur une matrice de 64 PEs, les PEs du bord sont reliés à leurs voisins par rebouclage: le processeur 63 est connecté aux PEs 0, 62, 7 et 55. Ceci permet de travailler sur un *tore* de processeurs, ce qui peut être intéressant en traitement d'images (si la taille de l'image *colle* exactement le nombre de processeurs).

-un partitionnement de chaque PE: qui permet d'obtenir soit 2 PEs de 32 bits, soit 8 PEs de 8 bits à partir d'un même PE de 64 bits. Ceci donne lieu à une configuration de 512 PEs de 32 bits, ou 2K PEs de 8 bits. Cette technique de partitionnement permet d'obtenir une machine traitant des mots relativement petits, ce qui est fréquemment le cas en traitement d'images, tout en ayant une machine constituée de processeurs puissants (les PEs travaillent sur 64 bits). Contrairement à d'autres machines fonctionnant avec des PEs 1-bit, que nous allons décrire, et qui sont plus orientées traitement d'images, Illiac IV offre une plus grande puissance de calcul puisqu'une opération élémentaire difficile (multiplication flottante par exemple) se fait sur 64 bits par PE, et non sur 1 bit comme dans ces autres machines.

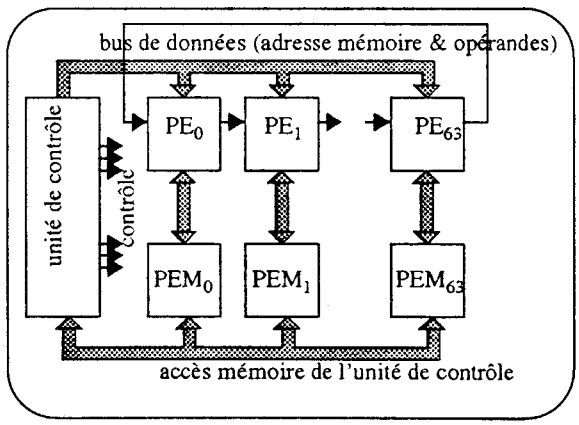


figure 2 structure d'une matrice de 64 PEs

Au niveau du contrôle global, on trouve:

-un envoi global des opérandes communs: les constantes et autres opérandes communs sont récupérés et stockés localement par l'unité de contrôle, et envoyés avec l'instruction de laquelle elles dépendent.

1. dans Illiac IV, un mot comporte 64 bits.

-un partitionnement matriciel: les 256 PEs de l'Illiac IV sont groupés en 4 matrices indépendantes de 8x8 PEs. On peut reconfigurer ces matrices en 2 tableaux de 128 PEs chacun, ou une grande matrice de 256 PEs. Pour chacune de ces configurations, le routage est fait de telle façon qu'il relie les PEs voisins. On obtient pour une matrice le schéma de la figure 2.

2.1.2 Processeur Elémentaire

Le PE d'Illiac IV est un processeur très puissant pour une machine SIMD. De fait, cette machine avait pour objectif le calcul intensif. Néanmoins, si la seule partie qui ait jamais été réalisée possède 64 processeurs de 64 bits (4K PExbits), elle reste comparable à des machines jugées plus raisonnables, parce que réalisées, et qui comme MPP ou maintenant la Connection Machine comportent respectivement 16K PEs 1-bit (16K PExbits) et 64K PEs 1-bit (64K PExbits).

On retrouve les éléments suivants:

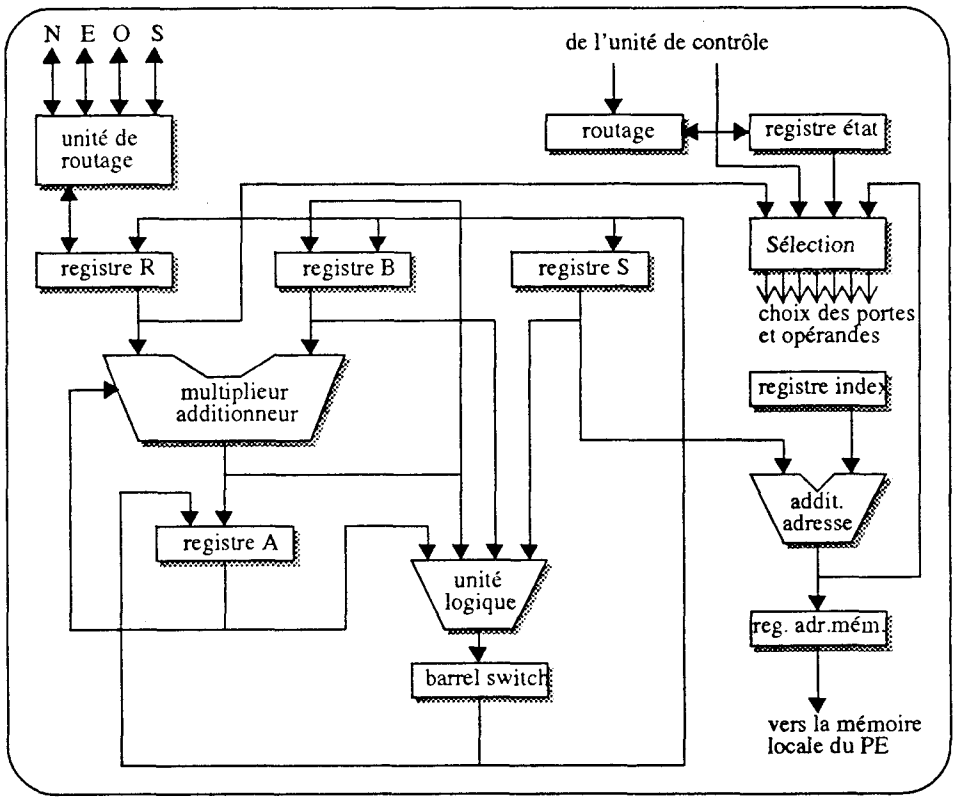


figure 3 Proc. Elém. de la machine ILLIAC IV

- quatre registres 64 bits (A,B,S et R) pour les opérandes et résultats. A sert d'accumulateur, B de registre opérande, R de registre multiplicande et de registre pour le routage, et S de registre général,

- un additionneur et multiplieur, une unité logique suivie d'un *barrel switch* pour les opérations de décalages arithmétiques et booléens,

- un registre index 16 bits, et son additionneur, pour les accès à la mémoire locale du PE,

- un registre d'état 8 bits qui contient les résultats des opérations ainsi que le bit d'état du PE: *enable/ disable.*

Aller plus en avant dans la description de cette machine n'apporterait pas beaucoup plus d'information, et d'ailleurs le temps relativement long qui nous sépare de cette machine a permis la mise au point d'autres plus performantes. Pour les applications d'Illiac IV, le lecteur peut se reporter à [KUCK68] qui présente des opérations matricielles, ainsi qu'un algorithme complet d'inversion de matrice. On trouve d'ailleurs dans cet article la méthode des tableaux décalés (*skewed matrice*), à laquelle nous ferons de nouveau allusion au chapitre 3, et qui reste employée dans les dispositions mémoire des supercalculateurs vectoriels.

2.2 CLIP IV

Les machines de la famille CLIP (*Cellular Logic Image Processor*) ont pour vocation le traitement logique des images, sur un modèle d'automates cellulaires qui s'inspire directement du modèle de Von Neuman. Les cellules sont disposées en une matrice 2D, de 96x96 processeurs. Chaque processeur comporte une unité logique 1-bit, à deux entrées et deux sorties. L'une des entrées correspond à l'un des pixels de l'image traitée ou à une variable locale (registre A). La seconde correspond à une sortie sélectionnée d'un des 8 voisins du processeur, ou le registre B. On remarque la combinaison préalable des 8 voisins ($\Sigma a_i v_i$ où v_i représente un voisin) qui permet d'accélérer certains traitements, morphologiques notamment [STAM75]. Pour les sorties, l'une d'entre elles est reliée aux voisins du processeur, l'autre est dirigée vers les registres d'entrée du processeur pour servir de nouvelle variable locale. Différents voisinages sont programmables: carré, hexagonal; de même le sous-ensemble des voisins d'un processeur.

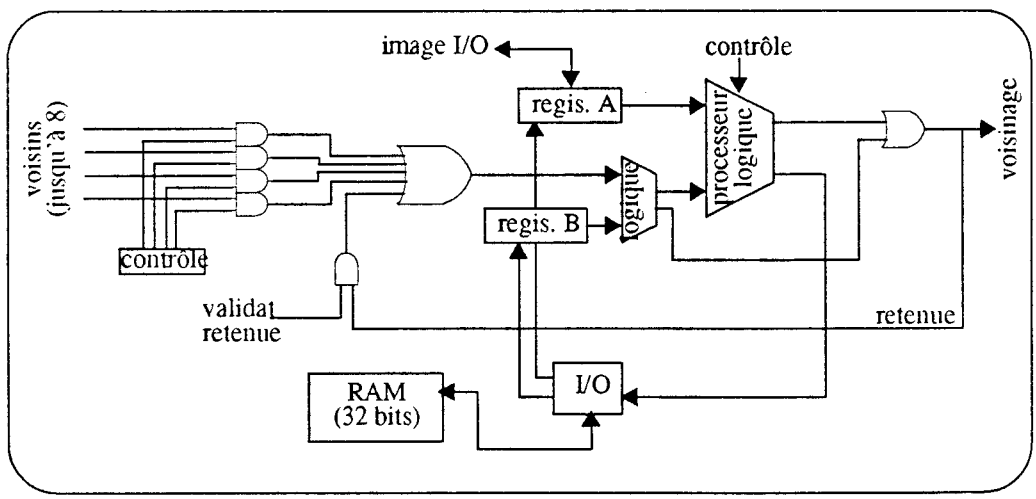


figure 4 processeur élémentaire de CLIP4

On trouve dans [STAM75] des exemples d'utilisation de cette machine, notamment en mode asynchrone où chaque PE met à jour sa ligne vers ses voisins de manière totalement asynchrone. On y trouve aussi un algorithme de tracé de segments, sans calcul de type Bresenham. Dans un espace à orientations fixées, l'algorithme crée une zone de pixels noirs située entre deux points; ensuite le squelette de la zone est extrait, qui donne le segment cherché.

CLIP appartient à une famille importante de machines cellulaires pour la reconnaissance de formes. Les algorithmes mis en oeuvre sur ces machines font le plus souvent référence aux méthodes morphologiques ou aux filtrages par convolution. Chaque image possède des pixels blancs ou noirs; pour traiter les images à niveaux de gris, l'algorithme travaille soit par tranches de 1 bit, soit par multi-seuillage: une image est seuillée plusieurs fois avec un seuil croissant.

L'intérêt de ces machines logiques cellulaires concerne d'abord la performance, puisque les unités logiques sont plus rapides que les unités arithmétiques; vient ensuite la densité d'intégration, qui permet d'atteindre des tailles de machines importantes. Enfin ces machines sont bien adaptées aux algorithmes de morphologies mathématiques et de convolutions. Que ces algorithmes de filtrage doivent être effectués rapidement et sur de grandes images, c'est un truisme qui devait être dit quelque part.

[PRES83] analyse l'existant dans le domaine des machines cellulaires logiques. Le lecteur pourra y retrouver la machine CLIP4, le Cytocomputer que nous avons déjà cité comme machine pipeline de traitement d'images, PICAP II construite autour d'un bus à haut débit sur lequel communiquent des processeurs spécialisés (filtrage, voisinage 3x3,...), et MPP que nous présentons ci-dessous.

2.3 Massively Parallel Processor

2.3.1 Description architecturale

La machine MPP [POTT83] possède 16K processeurs; Goodyear Aerospace l'a construite pour la NASA afin d'exploiter des images satellites très rapidement. La charge de travail envisagée pour une telle machine est de l'ordre du milliard d'opérations[1] par seconde, ce qui pour l'époque (contrat: 1979, livraison: 1982) est considérable. Le recours à une machine du type SIMD exploite le fait que les millions de pixels traités par image peuvent l'être simultanément, la tâche étant la même. La figure 5 présente un schéma synoptique de MPP.

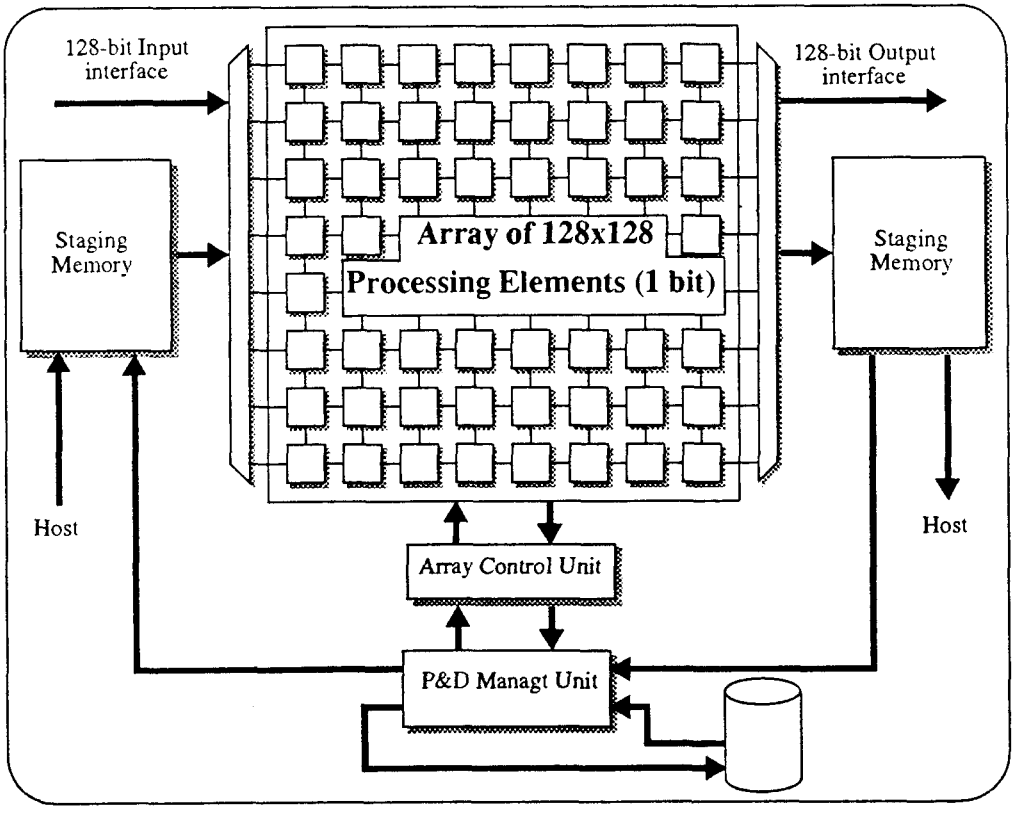


figure 5 Schéma synoptique du Massively Parallel Processor

1. La NASA lançait au début dès années 80 des satellites dont les capteurs d'images généraient de l'ordre de 10^{13} bits par jour. Les transformations effectuées sur ces images demandaient donc des puissances de l'ordre de 10^{9} à 10^{10} opérations par seconde.

Le MPP possède une matrice de 128x128 PEs fonctionnant sur 1 bit chacun, ainsi qu'une matrice de 4x128 PEs dans le cas où l'un des PEs ne fonctionnerait pas dans la matrice principale. La machine fonctionne à 10 Mhz, ce qui lui donne des possibilités réelles en calcul intensif: 6,5 Gadditions 8 bits par seconde, 0,43 Gadditions flottantes... Comparées aux machines vectorielles de l'époque, les performances de MPP étaient indiscutables, surtout pour une machine basée sur des PEs 1 bit.

2.3.2 *Topologie de routage*

Chaque PE dans la matrice communique avec ses quatre plus proches voisins. On retrouve une topologie de connexions identique que pour la machine Illiac IV, sur les bords de la matrice: les PEs d'un bord étant connectés avec ceux du bord opposé (topologie en cylindre). Cependant la connexion peut ici être décalée, offrant de fait une topologie de connexions en spirale. Celle-ci permet donc de créer une ligne continue de 16K processeurs pour les traitements linéaires (1D).

2.3.3 *Processeur Elémentaire*

Chaque PE [BATC80] possède six registres 1 bit (A,B,C,G,P,S), un registre à décalage avec une longueur programmable, une mémoire RAM, un bus de données, un additionneur et de la logique. L'horloge de base fonctionne à 100 ns.

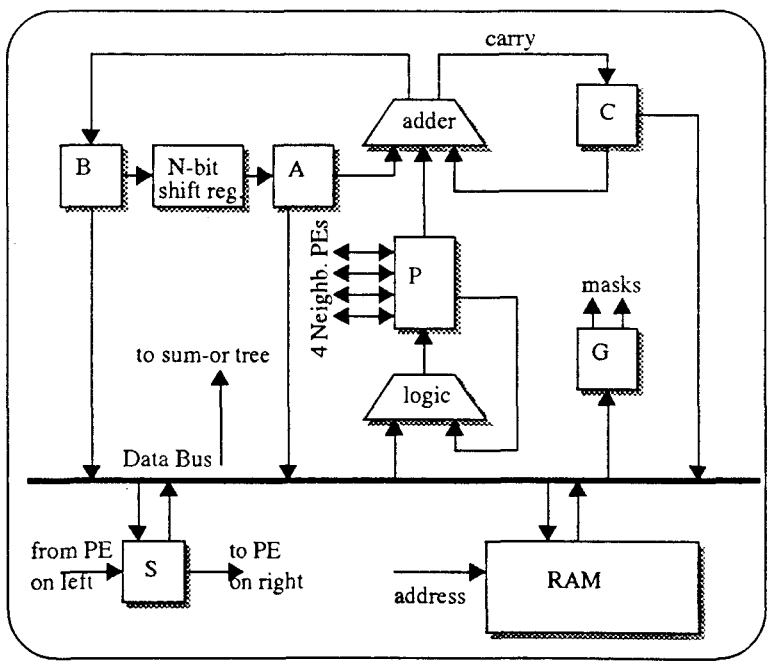


figure 6 Un Processeur Elémentaire de MPP

Le registre P est utilisé pour les séquences de routage entre le PE et ses 4 plus proches voisins. Chaque PE possède un additionneur complet 1 bit, avec retenue. Pour additionner deux nombres quelconques, on charge l'un de ceux-ci dans le registre A bit par bit, en commençant par le LSB (Least Significant Bit); l'autre nombre est chargé dans le registre P, bit par bit aussi; à chaque addition 1 bit, la retenue est stockée dans C et la somme dans B. Cette somme peut alors être stockée dans la RAM locale, ou dans le registre à décalage de taille variable N. Pour effectuer une soustraction, l'unité logique permet de modifier le contenu de P (complément à 1 par exemple).

La multiplication est une série d'additions, où le produit partiel est réinséré dans le processus itératif d'additions par le biais du registre à décalage et des registres A et B. Les multiples appropriés du multiplicande sont générés dans le registre P et ajoutés au produit partiel tandis qu'il circule par l'additionneur. De la même façon, ce PE permet d'effectuer des divisions et des opérations en virgules flottantes. Bien que ces opérations s'effectuent par tranches successives de 1 bit, le MPP atteint des puissances effectives de calcul importantes dues à l'exécution massivement parallèle des opérations (470 MFlops pour des additions 32-bits en virgule flottante).

Le registre G permet de masquer l'exécution par l'un des PEs, d'une opération demandée à tous les PEs simultanément. Ainsi tous les PEs dont G=1 n'effectuent pas les opérations requises par l'ACU (Array Control Unit). Les opérations logique et de routage sont masquées indépendamment de l'opération arithmétique sur un PE. Quant à la RAM, d'une capacité de 1K bits, elle fait usage de mémoire locale pour le PE.

Pour terminer cette description, les états de chacun des bus internes des PEs, non masqués, sont combinés dans un arbre d'éléments OU, nommé l'arbre de *somme-ou*. Cette sortie est gérée par l'ACU pour certains tests globaux, tels que la recherche de la valeur maximale ou minimale après une opération distribuée parmi tous les PEs. Quant au registre S, son rôle consiste à transmettre des données depuis l'entrée de la matrice jusqu'à sa sortie. Tandis que chaque PE exécute des opérations à l'aide de leur mémoire locale, les données sont décalées de gauche à droite dans toute la matrice *via* le registre S. Nous verrons que cette technique, qui permet d'atteindre des débits de transfert entrée-sortie importants (160 Moctets/s), se retrouve dans les processeurs systoliques réputés gourmands pour ce type d'échange transparent.

2.3.4 Mémoire étagée

Ce qui est intéressant dans cette machine, c'est l'accès mémoire par bloc (*staging memory*) qui pour une adresse donnée retourne un bloc de 16K bits. Dans cette mémoire, les données sont agencées de telle façon qu'elles facilitent l'accès à un bloc de 16K bits quelque soit la disposition des pixels dans l'image complète. Ainsi, si *k* est l'adresse passée à la mémoire, les données suivantes sont chargées dans la matrice de PEs:

k, k+n, k+2n, ...k+127n,
k+m, k+m+n, k+m+2n,.....k+m+127n,

k+127m, ...k+127m+127n

Nous reviendrons sur l'utilité d'une telle méthode de stockage dans le domaine du traitement d'images. Néanmoins, pour ne pas en rester là, il faut préciser que cette technique fut employée dans MPP pour pallier la répartition de l'information dans les images satellites. En effet, les images satellites sont enregistrées dans l'ordre pixel consécutif (1er pixel de la 1ère ligne, suivi du 2nd pixel de la 1ère ligne,...), tandis que les applications peuvent requérir un arrangement par bande entrelacée (toutes les bandes spectrales du premier pixel consécutives) ou par bande séquentielle (tous les pixels de la première bande spectrale, suivis de tous le pixels de la seconde bande spectrale,...).

2.3.5 *Application au traitement d'images*

Voyons maintenant sur deux exemples, comment une telle machine se programme. Pour cela, nous allons décrire deux algorithmes souvent utilisés en traitement d'images bas-niveau, tels que la détection de contours ou de régions, et les convolutions 2D. Ces algorithmes n'ont rien d'originaux, mais permettent de se faire une idée d'abord de la puissance de description parallèle de la machine MPP, et des machines SIMD en général, mais aussi de certaines de ses faiblesses, révélées par une analyse fine de la complexité du second algorithme présenté.

Convolution

Soit une matrice M de 128x128 éléments, sur laquelle on désire effectuer une convolution 3x3 avec une matrice W de poids: W(3,3) est stockée dans la mémoire du contrôleur, l'ACU. La matrice M(128,128) est projetée sur les PEs, de telle manière que le PE(x,y) se voit affecter la valeur M(x,y). Alors l'algorithme de convolution qui génère la matrice résultat CV(128,128) est le suivant:

```
CV($,$) = 0
Pour i= -1 à 1 faire
        Pour j= -1 à 1 faire
                CV($,$) = CV($,$) + W(i,j)*M($+i,$+j)
        finpour
finpour
```

où le parallélisme est indiqué par la notation $, où M($+1) indique que chacun des PE doit extraire sa valeur de M et l'a transmettre à son voisin de gauche: le processeur PE(x,y) utilisant la valeur M(x+1,y) qui est stockée dans la mémoire locale de son voisin de droite, le PE(x+1,y). La puissance d'une telle machine apparaît évidemment à la lecture du programme, somme toute très simple.

Extraction de contours

Le gradient est l'un des opérateurs de contour les plus anciens et connus [ROBE65]. Nous le présentons ici pour l'implantation de l'algorithme sur MPP, et parce que nous en parlerons à nouveau au chapitre 2. Pour une image $f(x,y)$, la magnitude du gradient $s(x,y)$ et sa direction $\theta(x,y)$ s'expriment de la façon suivante [BALL82]:

$$\begin{cases} s(x,y) = \sqrt{\Delta_x^2 + \Delta_y^2} \\ \theta(x,y) = \operatorname{atan}\left(\dfrac{\Delta_y}{\Delta_x}\right) \end{cases}$$

où

$$\begin{cases} \Delta_x = f(x+n,y) + f(x,y) \\ \Delta_y = f(x,y+n) + f(x,y) \end{cases}$$

Le plus souvent, n est petit; son choix est tel que le gradient obtenu est une bonne approximation des

changements locaux en *(x,y)*. L'intérêt du gradient de Sobel, qui se calcule sur un voisinage 3x3, vient de ce qu'il diminue l'influence du bruit par le calcul d'une moyenne sur un voisinage plus important:

Δ_x

-1	0	1
-2	0	2
-1	0	1

Δ_y

1	2	1
0	0	0
-1	-2	-1

pour le voisinage ci-dessous, on obtient:

a	b	c
d	*pix*	e
f	g	h

$$\begin{cases} \Delta_x = \frac{1}{4} \cdot [\,(c+2.e+h) - (a+2.d+f)\,] \\ \Delta_y = \frac{1}{4} \cdot [\,(a+2.b+c) - (f+2.g+h)\,] \end{cases}$$

L'implantation de l'algorithme est extraite de [ROSE88]; il fonctionne sur une image de taille *nxn* pixels, où *n=128*, c'est à dire la taille de MPP. Il suppose que chaque PE sait s'il se trouve sur le bord de la matrice de PEs, ce qui signifie que l'une de ses coordonnées *lig* et *col* au moins égale soit *0* soit *n-1*. Les pixels de l'image sont codés en niveau de gris, sur *B* bits.

Voici les différentes phases de l'algorithme:

(1) Chaque PE, qui ne se trouve pas sur la première ou la dernière ligne de la matrice de PEs, récupère les valeurs de niveau de gris des PEs voisins au dessus et en dessous de lui. Soit *u* et *w* ces valeurs respectives, et soit *v* la valeur originale du PE, chaque PE calcule donc *u+2v+w*. Si les niveaux de gris ont *B* bits, le temps requis pour ce calcul est:

$2.B.t_s$	pour la copie entre PEs,
$B.t_+$	pour le calcul *u+w*,
$(B+1).t_+$	pour le calcul *(u+w)+2.v*.

(2) Chaque PE, qui ne se trouve pas sur la première ou la dernière colonne de la matrice de PEs, récupère les valeurs u+2v+w des PEs de gauche et de droite, et les soustrait pour obtenir Δ_x. Ceci requiert $2(B+2)t_s + (B+2)t_-$.

(1'-2') Δ_y est calculé de la même façon.

(3) Chaque PE n'appartenant pas au bord peut déterminer si son pixel appartient ou non à un contour, en calculant la magnitude $s^2(x,y)$ et en la comparant à un seuil T^2. Ceci requiert:

$(B+3)^2.t_x$	calcul de Δ_x^2 et Δ_y^2,
$2.(B+3).t_+$	addition de $\Delta_x^2 + \Delta_y^2 = s^2(x,y)$,
$[2.(B+3)+1].t_e$	envoi global de T^2,
$[2.(B+3)+1].t_c$	comparaison entre s^2 et T^2.

(4) Chaque PE calcule $tan\theta$ en divisant Δy par Δx; ensuite, pour calculer l'angle θ, on utilise une méthode par *lookup table*. L'unité de contrôle envoie une succession de valeurs $(tan\theta, \theta)$ aux PEs. Chaque PE compare la valeur envoyée de $tan\theta$ à celle qu'il a calculée. Si elles sont égales, il stocke la valeur de θ correspondante. Si les valeurs de θ et $tan\theta$ sont sur b bits, alors le temps requis est:

$(B+3)^2.t_/$ calcul de $\theta(x,y)$,

et pour chaque couple de la *lookup table*:

$2.b.t_e$ transfert du couple,
$b.t_c$ comparaison.

[ROSE88] donne les temps correspondants sur la machine MPP à ceux utilisés ci-dessus. On rappelle que le temps de cycle de MPP est de *100ns*.

temps	nbre de cycles par bit
t_s	3
t_e	2
t_c	3
t_+, t_-	3

2.4 Connection Machine

2.4.1 Introduction

La *Connection machine* fut la première *vraie* machine SIMD massivement parallèle, 64K processeurs, ce que son écho médiatique dans le microcosme informatique des années 1985-88 ne démentit pas. Comme le remarque [HILL85], concepteur de la machine, la difficulté technique tient dans la conception du réseau d'interconnexion de ces 64K processeurs. Héritant des connaissances acquises par les projets réussis de MPP (16 K processeurs 1 bit) ou DAP[1] (4 K PEs 1 bit), *la* machine des années 1980 se devait non seulement d'accroître le nombre de PEs utilisés, mais encore d'offrir une communication entre PEs plus élaborée.

L'interconnexion des processeurs est justement considéré comme "le problème technique le plus important dans la conception de la machine à connexions" (sic [HILL85], page *64* traduction française). Cette difficulté provient de l'impossibilité dans laquelle se trouve le concepteur de connecter directement chaque paire de PE par une matrice de commutation (*crossbar*) quand le nombre de PEs, N, devient grand. La complexité matérielle de cette matrice évolue comme le carré de N ! Le réseau d'interconnexion est donc un réseau à commutation par paquets construit sur la topologie des réseaux n-cubes booléens.

1. Digital Array Processor, est une machine sensiblement voisine de CLIP IV, si ce n'est que la communication est réduite aux quatre voisins [DANI81]. Comme MPP, cette machine se retrouve souvent dans la littérature, utilisée comme plate-forme pour tester des algorithmes SIMD de traitement d'images (convolution, rotation,...).

Avant de détailler la machine, il convient de préciser qu'il en existe deux modèles: CM1 et CM2. La CM1 contient 64K PEs 1 bit, accédant chacun une mémoire locale de 4k bits. Tous les processeurs exécutent la même instruction, générée par un micro-contrôleur connecté à un frontal. Les PEs sont regroupés par 16, et forment une unité d'exécution avec routeur et mémoires locals. Elle se présente sous la forme d'un cube de 1,3 mètre de côté. Sa puissance est de 1000 MIPS, sur des additions 32 bits.

Les opérations arithmétiques s'effectuent de manière sérielle sur 1 bit, en 750ns. La première CM1 fut livrée en été 1986 au MIT [TUCK88]. La CM2 diffère relativement peu de la CM1. Le nombre de processeurs est le même, mais ceux-ci accèdent dorénavant 64K bits de mémoire locale. Chaque groupe de 32 PEs possède une unité flottante 32 bits (en option) qui permet d'accélérer les calculs flottants. Les performances en opérations entières restent stables: 2500 MIPS en additions 32 bits. Par contre, grâce à l'adjonction d'unités flottantes, la CM2 supporte 20 GFLOPS... au mieux.

2.4.2 Organisation générale

La CM1 est découpée en 4 parties de 16K PEs, contrôlées chacune par un séquenceur propre. Un réseau *cross-point switch* permet à ces 4 séquenceurs de recevoir des *macro-instructions* d'une des 4 lignes de communications rapides avec le frontal (une machine VAX sous ULTRIX, par exemple). Ces *macro-instructions* sont décodées par le séquenceur, qui génère alors des *nano-instructions* en directions des contrôleurs locaux des PEs. Ce décodage, similaire à un micro-code, permet au frontal de communiquer à une vitesse plus faible que la distribution réelle des instructions SIMD aux PEs. On retrouve évidemment une FIFO, qui contient plusieurs *macro-instructions* en attente d'exécution (cf. MPP,ILLIAC IV,...).

Si le coeur du système est cette unité de calcul parallèle qui comporte 16K processeurs, la CM1 n'en dispose pas moins d'un système de communication entrée-sortie performant, qui permet l'accès à une mémoire de masse importante (disques) et à un contrôleur graphique. Chaque disque offre un espace de stockage de cinq à dix GOctets, et une vitesse de transfert de 40 MO/s. La mise en parallèle de huit disques permet d'offrir à la CM une vitesse de transfert de 320 MO/s, pour les chargements des mémoires locales des PEs.

Enfin pour la partie graphique, une mémoire de trame de 1280x1024 pixels sur 24 bits est prévue. Ce module mémoire est directement connecté à la machine, ce qui permet des vitesses de transfert de un Gbits par seconde [TUCK88].

2.4.3 Le processeur élémentaire

Le composant de base est un circuit VLSI *full-custom*, qui comporte 16 cellules processeur, une unité de contrôle et une unité de routage. L'unité de contrôle décode les *nano-instructions* délivrées à chaque circuit, et génère les signaux de contrôle correspondant pour les 16 PEs et l'unité de routage. Le routeur se charge du transport des messages entre chaque circuit, à l'aide de leur adresses. Ce routeur comporte trois parties: un injecteur, qui transmet des messages dans le réseau, une unité de traitement qui se charge du suivi continu des messages, et l'éjecteur qui reçoit et délivre les messages à l'élément de calcul approprié [HILL85].

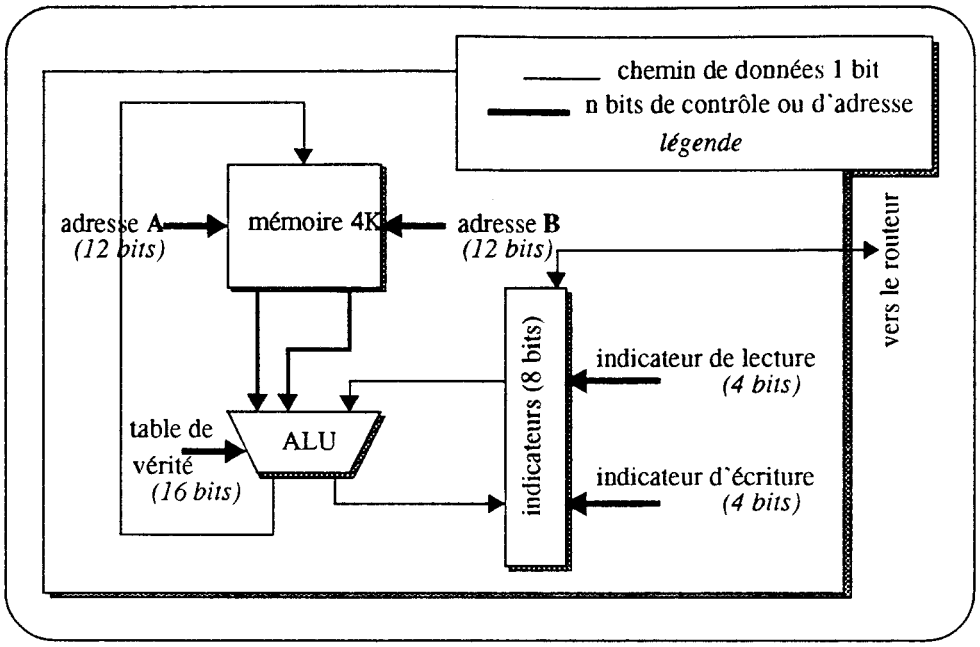


figure 7 Elément de calcul de la Connection Machine

Pour les communications entre PEs d'un même circuit, ceux-ci sont connectés sur une grille à quatre voisins (Nord, Est, Sud, Ouest) indépendamment du routeur. Ce schéma de connexion facilite les communications locales, comme pour les machines vues jusqu'ici. La figure 7 présente un schéma du PE 1 bit de la CM. Les fonctionnalités de celui-ci ne se sont guère modifiées par rapport à MPP, ou CLIP IV. Nous notons seulement la disparition des registres (à part le registre indicateur, *flag register*), et un fonctionnement en trois cycles du PE: choix de l'opérande A, choix de l'opérande B, choix d'un indicateur; puis évaluation du résultat à partir de ces entrées et de la fonction sélectionnée dans l'ALU; et enfin rangement dans la mémoire d'un des résultats, le second étant stocké comme nouvel indicateur. Ces trois cycles s'effectuent en 750ns sur CM1. Le lecteur intéressé par l'architecture et le fonctionnement précis de ce PE trouvera dans [HILL85] ou [TUCK88] les réponses attendues.

2.4.4 Le modèle de communication

Chaque bloc de 16 PEs possède son propre routeur; cela fait 4K routeurs pour la CM entière. La topologie d'interconnexion est un *n*-cube, où n=12. Dans ce réseau chaque routeur est identifié par une adresse sur 12 bits, entre 0 et 4095. Le routeur d'adresse *i* est connecté au routeur d'adresse *j* si et seulement si $|i-j|=2^k$ où *k* est entier.

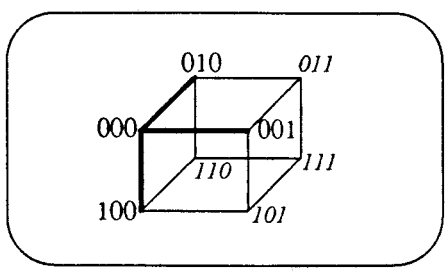


figure 8 les routeurs et leur adresse respective, pour un réseau *n*-cube avec n=3

Les routeurs sont connectés selon la $k^{ième}$ dimension d'un n-cube, c'est à dire un espace euclidien de dimension n (cf. figure 8). Comme deux adresses ne peuvent différer entre elles sur plus de 12 bits, on peut atteindre un sommet quelconque du cube en parcourant au plus 12 sommets (sur les 2^{12} que comptent le réseau). Enfin, le fait de masquer certains routeurs du réseau entraîne une nouvelle configuration de celui-ci en plusieurs petits réseaux distincts.

Le transfert de messages dans le réseau se fait par adresse relative à l'adresse du routeur qui contient actuellement le message. Si cette adresse est égale à 0, alors le message est destiné à l'un des PEs attachés au routeur qui possède le message. Si l'adresse relative diffère de l'adresse 0 de k bits, alors le message doit encore être transmis k fois entre routeurs voisins dans l'hypercube. Chaque routeur possède des mémoires tampon, dans lesquelles il stocke provisoirement les messages avant de les transmettre à qui de droit. Au cas où ses mémoires tampon sont pleines, il n'accepte plus de messages. [HILL85] remarque que le débit moyen des échanges sur le réseau, en régime stationnaire, du prototype de la CM est de 10 Gbits/s, pour un temps de cycle de 500ns! Quant à [TUCK88], il rapporte un algorithme de tri de 64K clefs de 32 bits chacune, effectué en 30ms!

2.4.5 Applications

La CM présente du point de vue des communications un avantage considérable sur les machines SIMD à communication réduite aux quatre ou huit premiers voisins. Ainsi [TUCK88] présente la mise en oeuvre d'un algorithme de placement de cellules standard sur un circuit VLSI, à l'instar de ce que propose [KIRK83]. Kirkpatrick, dont l'article nous a séduit par sa présentation claire et complète du recuit simulé, explique l'utilité de cette méthode, héritée de la mécanique quantique et connue pour son efficacité dans la résolution du problème du voyageur de commerce. Le recuit simulé produit en effet de bons résultats, solutions proches de la meilleure, mais en des temps malheureusement trop importants quand il est développé sur des calculateurs conventionnels: de l'ordre de 100 à 300 heures. La CM propose une solution en 2 heures! Il faut remarquer que cet algorithme se présente sous une forme parallèle naturelle; les gains annoncés le corroborent.

Une autre application, toujours pour l'électronique, concerne la simulation électrique d'un circuit composé de résistances, capacités, transistors,... A l'aide de la loi de Kirchoff, qui donne la somme des courants entrant dans un noeud du circuit égale à la somme des courants sortants, cette simulation se ramène à la résolution d'un système d'équations différentielles non linéaires par un algorithme de relaxation de Gauss-Jacobi [TUCK88]. Des simulations de circuits de plus de 100 000 transistors se font en quelques heures...

Appliquée à la vision, cette machine permet la recherche *associative* de formes, de manière parallèle. A chaque objet connu de la machine est associé un ensemble de caractéristiques du type segments+coin. L'image est alors chargée sur la machine, 1 pixel pour 1 PE. Ensuite les segments et leurs coins sont détectés, puis regroupés jusqu'à obtention d'une ou plusieurs des caractéristiques d'une forme connue. Chaque fois qu'une ressemblance est repérée entre une forme et un certain nombre de ses caractéristiques, une instance de l'objet supposé est créée. Ensuite cette instance peut être testée: la machine vérifie si l'objet créé existe ou non dans l'image par une sorte d'appariement dans l'espace 2D [TUCK88].

Dans le même esprit, [TUCK88] rapporte une application de vision stéréoscopique. L'objectif est de déterminer le relief d'une scène à partir de deux clichés 2D pris de deux rétines voisines (gauche et droite). Ce projet de vision binoculaire se retrouve chez [FAUG88], mais avec une approche "calculateur dédié" dont nous reparlerons. Celui-ci remarque que la CM présente deux inconvénients majeurs: son coût, de l'ordre de vingt millions de francs, et surtout la *quasi*-impossibilité de la faire fonctionner au

rythme vidéo, *i.e.* vingt-cinq images par seconde, pour la simple question du chargement des mémoires locales à chaque PE. D'ailleurs [TUCK88] accrédite lui-même cette thèse, en annonçant le temps d'exécution de cette opération sur un couple d'images stéréo de 256x256 pixels: deux secondes.

Peu spectaculaires, si ce n'est en rapidité relative de fonctionnement, ces développements d'algorithmes de traitement d'images ne doivent pas nous voiler l'originalité essentielle de la CM. Cet originalité se situe à l'intersection critique du traitement massif et de la communication élaborée. Il est justement temps de clore là le survol des machines SIMD, et d'analyser leurs évolutions à venir.

2.5 Critique générale des machines SIMD

A la lecture des pages précédentes, le concept SIMD semble solidement implanté dans le domaine du traitement d'images. Ce constat jaillit pour les traitements à caractère local, c'est à dire ceux qui ne demandent qu'une communication souvent réduite aux voisins immédiats de chaque unité de traitement. L'architecture de ces processeurs élémentaires est si simple, unité à 1 bit, qu'elle offre tous les gages d'une intégration de plus en plus importante dans le silicium, à un moment où la technologie passe du VLSI à l'ULSI. Quant au modèle à quatre voisins, mais aussi à huit, il n'induit pas une augmentation de complexité dans le dessin de ces processeurs.

Toutefois ces machines sont relativement coûteuses. Ce coût est d'ailleurs lié étroitement au nombre de processeurs de la machine réalisée, et donc aux performances directes de celle-ci. Le parallélisme est certes intrinsèque, mais pour des problèmes de taille bien définie (le nombre de PEs en général). S'il faut traiter un problème de taille plus grand que celle de la machine, là commencent les difficultés et surviennent les pertes de temps.

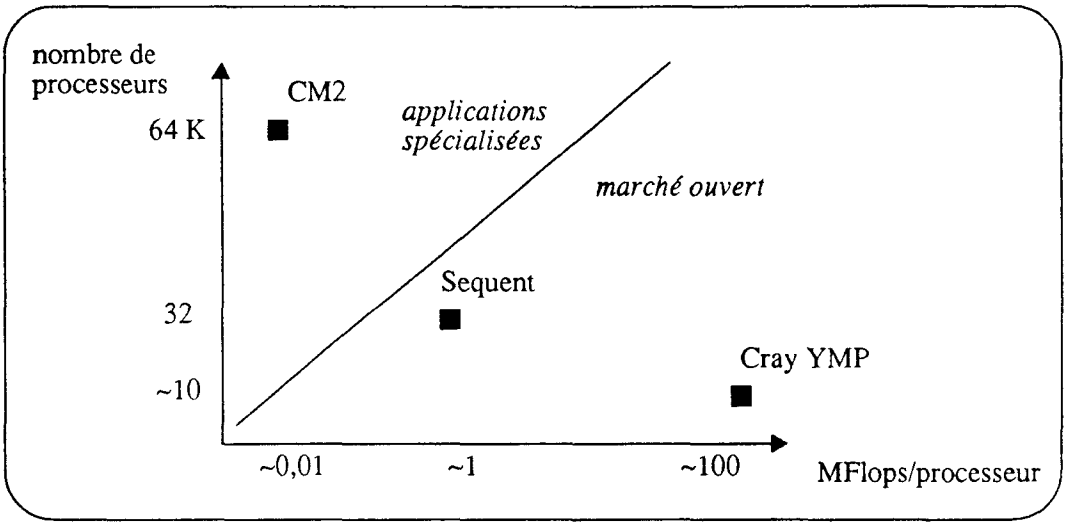
La Connection Machine est la première machine SIMD qui, tout en fonctionnant -puisqu'on ne saurait discuter honnêtement d'architectures s'il n'y avait aucune réalisation réelle de celles-ci-, offre un modèle de communication plus élaboré que ses prédécesseurs. Le projet ELSA, que nous détaillerons par la suite, se donne comme objectif la réalisation d'un circuit VLSI dans lequel seraient intégrés un nombre important de processeurs 1 bit à communication réduite aux quatre premiers voisins. Nous voyons ici la preuve d'une scission à terme de la famille des machines SIMD. Cette division s'opérera certainement parce que les difficultés technologiques apparaissent et rendent nécessaire des choix fondamentaux.

Ainsi le concepteur d'aujourd'hui doit choisir son camp: ELSA en est un, la CM un autre. Le premier se caractérise par une approche cellulaire évidente. La technique permet d'envisager les automates cellulaires dont rêvait déjà Von Neuman, par cet accroissement prodigieux des densités d'intégration de processeurs simples pour lesquels les communications s'adaptent parfaitement à l'espace physique, matériel qui les supporte: communications aux quatre voisins. A l'opposé, cette architecture pénalise certains traitements à communications distribuées dans l'espace. Ces traitements devront s'effectuer sur des machines qui possèdent un réseau d'interconnexion évolué, capable de faire communiquer bon nombre de processeurs souvent voisins, mais parfois distants. La CM a ouvert cette voie.

Nonobstant ces contraintes réelles, le concept SIMD reste avant tout un formidable outil de calcul, puisqu'il bénéficie à la fois du côté séduisant si caractéristique d'une réalisation à haut parallélisme, et du côté rassurant qu'offre la théorie du calcul qui lui est attachée. Contrairement aux machines MIMD, dont on ne connaît jamais complètement les inter-blocages possibles, celles SIMD semblent plus *domestiques*. Toutefois, pour les machines au réseau d'interconnexion élaboré, il tarde de caractériser finement le rôle réel des échanges, et de montrer comment ceux-ci réalisent une partie importante du calcul, comme le remarque [HILL85].

3 Machines à contrôle distribué

Dans cette partie, nous nous intéressons aux machines MIMD et leurs dérivées. Après la différence que nous avons constatée en introduction entre les machines à mémoire partagée et celles à communication par messages, une comparaison avec les machines précédentes s'impose. En effet, après avoir étudié les machines SIMD, on peut tenter de situer les puissances de ces machines par rapport aux machines MIMD. [PATT90] propose le graphe suivant:



Ce diagramme reflète très justement que les modèles MIMD les plus répandus correspondent à une nécessité économique qu'expliquent les machines IBM3090, VAX9000, ou les séries des CRAY multiprocesseurs: CRAY2, puis CRAY XMP, et enfin CRAY YMP. Si tous les efforts actuels portent sur la structure de la mémoire, ceci peut se comprendre par une limite qu'atteignent à la fois les technologies d'intégration, mais surtout les gains qu'offre l'architecture pipeline vectorielle. La rapidité d'accès mémoire est alors le critère *quasi*-absolu d'efficacité d'une machine multi-processeurs.

Pour la vision, nous avons retenu deux types de machines MIMD: les machines M-SIMD reconfigurables, et les machines pyramidales. Très proches l'une de l'autre, elles insuffleront certainement un regain d'intérêt au niveau théorique pour ces machines à contrôle distribué.

3.1 PASM, une machine multi-microprocesseurs reconfigurable

Les machines que nous avons présentées appartiennent pour l'instant à un modèle bien défini, SIMD en l'occurence. Seule la machine Illiac IV devait être construite sur un modèle M-SIMD, c'est à dire une ou plusieurs machines SIMD indépendantes (cf. les machines vectorielles multiprocesseurs, si on classe leurs processeurs dans les machines SIMD). La machine PASM est un exemple d'architecture multi-microprocesseurs, appliquée au traitement d'images. L'intérêt de cette machine est d'autoriser les reconfigurations entre une machine SIMD pour les traitements à caractère local, et une machine MIMD pour les traitements à caractère global comme la reconnaissance de formes avec multiples copies de l'image et des formes sur chaque processeur.

La machine PASM (PArtitionable-Simd-Mimd system) évolue dans un espace de reconfigurabilité à trois dimensions. En effet, constituée d'un nombre importants de PEs, ceux-ci peuvent être regroupés entre eux: chaque groupe forme une sous-machine (*partition*), qui communiquent par un réseau d'inter-

connexion modifiable (*connexions entre PEs*), et dont le fonctionnement au niveau de la sous-machine est soit MIMD soit SIMD (*mode de parallélisme*). Construire une telle machine paraît aujourd'hui raisonnable. La figure 9 présente une vue générale de cette machine, dont nous allons étudier les aspects particuliers.

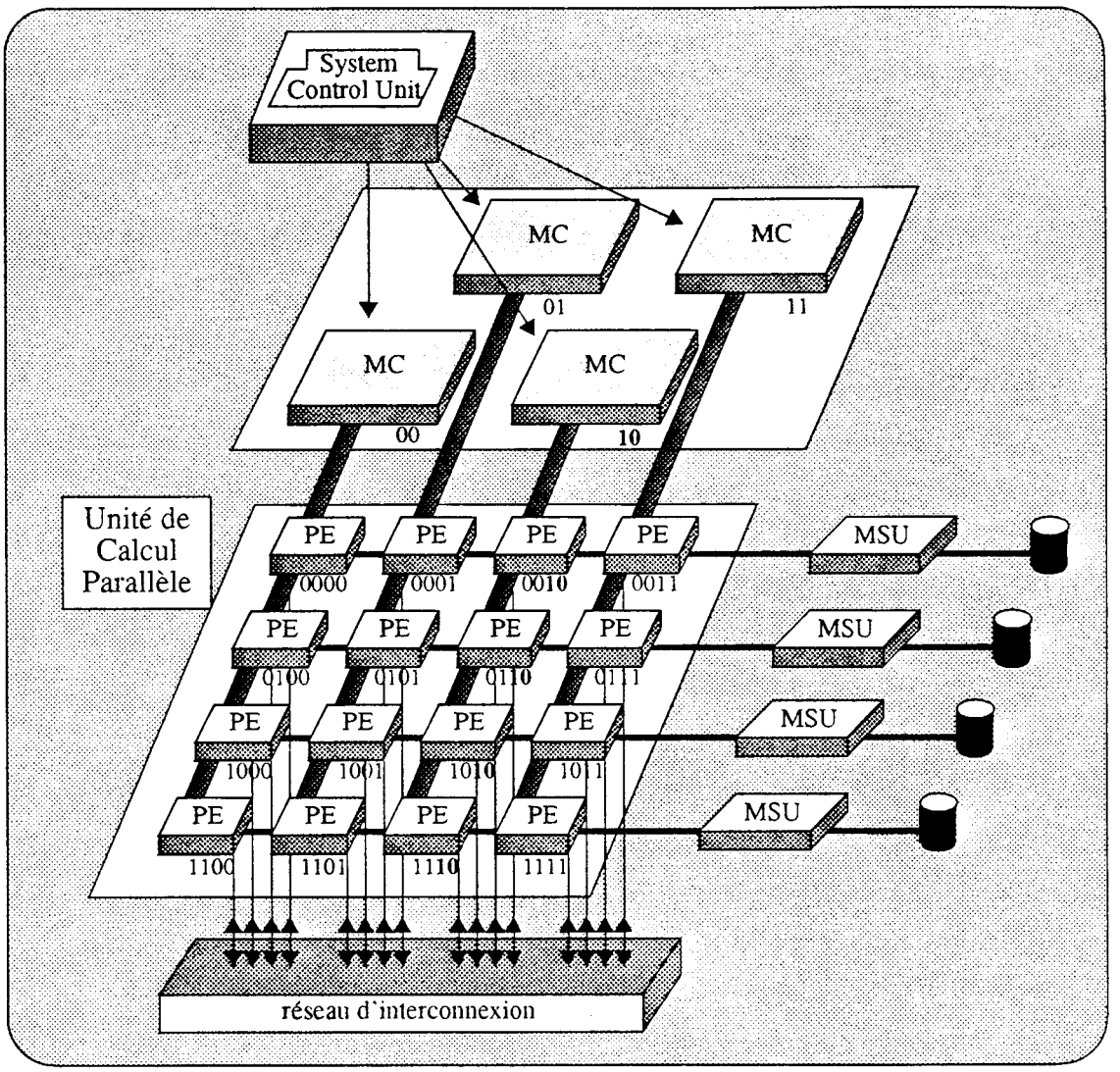


figure 9 la machine PASM, pour N=16 et Q=4

3.1.1 Les Micro-Contrôleurs

Les micro-contrôleurs, MCs, sont les unités de contrôle nécessaires à un fonctionnement M-SIMD. Il y a $Q=2^q$ MCs, qui possèdent chacun une adresse physique de 0 à Q-1. Chaque MC contrôle un groupe fixé de N/Q PEs. Ce MC et les PEs qui lui sont associés forment un *groupe MC*. Ce groupe possède l'adresse physique du MC; chacun des PEs possède à son tour une adresse, dont les q bits de poids faible sont les mêmes pour tous les PEs d'un groupe MC donné (cf. figure 9). Dans un système N=1024 processeurs, Q serait de l'ordre de 32. Le prototype de l'Université de Purdue possède N=16 processeurs, et Q égale 4.

En mode SIMD, chaque MC sert d'unité de contrôle aux Q PEs qu'il gère. L'unité mémoire jointe au MC lui permet de lire les instructions et de récupérer les opérandes globales, pour contrôler le flux

d'instructions SIMD. En mode MIMD, cette unité mémoire permet de stocker les instructions et les données nécessaires à la coordination des opérations de ces PEs, qu'assume le MC. De manière plus générale, [SIEG90-1] remarque que les instructions de contrôle et de flux sont effectuées par le MC, tandis que celles de traitement des données sont envoyées aux PEs. L'unité mémoire est doublée, pour permettre la préparation ou la terminaison d'un programme, pendant le déroulement d'un autre.

Une sous-machine est construite à partir d'un ou plusieurs groupes MC. Chaque sous-machine contient donc $R=2^r$ groupes MC, où $0 \leq r \leq q$. Une sous-machine comporte les $R.N/Q$ PEs dont les q-r bits de poids faible correspondent. La règle de reconfigurabilité de PASM précise en effet que les adresses de tous les PEs d'une sous-machine de taille 2^p correspondent sur les n-p bits de poids faible. Chaque MC possède dans sa sous-machine une adresse logique, qui est donnée par les r bits de poids fort de son adresse physique. De même, chaque PE possède une adresse logique dans sa sous-machine, qui est donnée par les $r+n-q$ bits de poids fort de son adresse physique. L'avantage de cette structure fixe de connexion PE-MC est son coût. Son inconvénient est évidemment la taille d'une sous-machine quelconque: qui doit être en puissance de 2, et ne permet donc pas une utilisation efficace de tous les PEs pour certaines configurations.

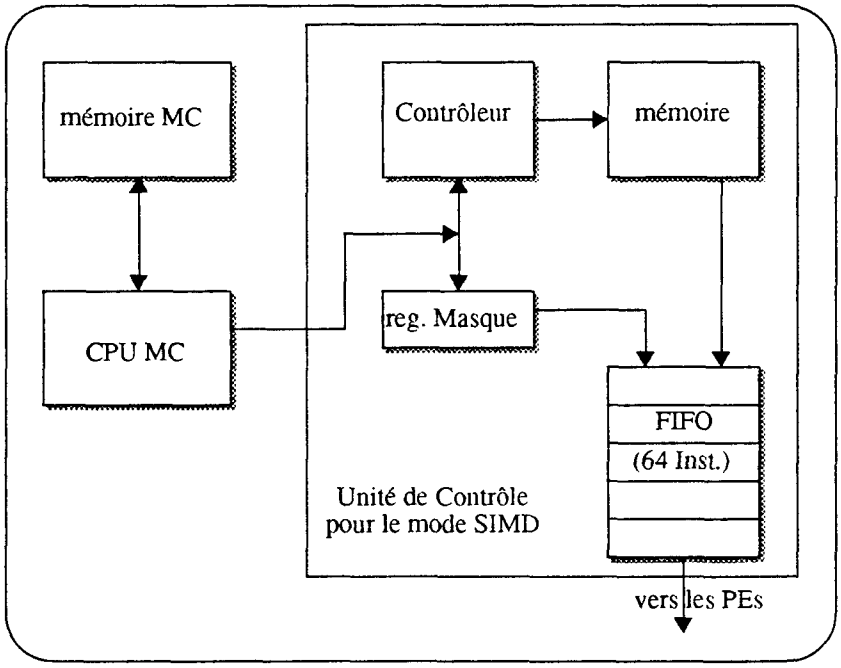


figure 10 le MC et son UC pour le mode SIMD

En mode SIMD, les R MCs d'une sous-machine doivent utiliser le même jeu d'instructions, c'est à dire posséder chacun la copie du programme à effectuer. Les PEs doivent être synchronisés. Une simple gestion câblée de l'envoie de l'instruction par les MCs vers les PEs, permet de synchroniser les PEs entre eux.

On retrouve sur PASM le mécanisme de masquage des PEs pour l'exécution des conditions. Ce masque est à deux niveaux: soit on masque un ensemble de PEs par leur adresse respective, *PE-address mask* [SIEG90-1] -intéressant pour bloquer un groupe de PEs-, soit on masque de manière locale un PE, *Data Conditional mask* -pour les conditions dépendant des données du PE, quelque soit son groupe-. Quant au test global d'un résultat sur tous les PEs d'une sous-machine SIMD, il est géré par les MCs de manière rapide.

Dans la machine prototype réalisée, chaque MC est construit autour d'un microprocesseur MC68000, avec son propre bus privé et 2MOctets de RAM dynamique. Les connexions entre SCU, MMS et MCs distants sont réalisées à l'aide de simples ports parallèles, dédiés au transfert d'informations de contrôle. La CPU saisie dans sa mémoire les instructions à exécuter, et contrôle le flot de celles-ci dans le cas de branchements conditionnels ou de boucles. Elle gère une unité de contrôle qui sert d'interface entre cette CPU et les PEs (cf. figure 10). Cette UC possède une FIFO de 64 instructions, telle qu'on en retrouve dans d'autres machines SIMD (Illiac IV, Connection Machine).

3.1.2 L'unité de Calcul Parallèle

Cette unité contient $N=2^n$ PEs et un réseau d'interconnexion. Deux unités mémoire sont associées à chaque PE, pour permettre le recouvrement de l'exécution d'un programme (mémoire locale du PE) et les opérations d'entrées-sorties (chargement d'un nouveau programme). Le réseau d'interconnexion permet aux PEs de communiquer entre eux. Dans la machine prototype existante, les PEs sont des processeurs MC68000 de chez Motorola. L'utilisation de processeurs dédiés ne remet pas en cause le fonctionnement de la machine. Deux caractéristiques de cette unité de calcul parallèle méritent d'être révélées: d'une part le mécanisme d'adressage de chacun des PEs, notamment la différenciation des modes MIMD et SIMD, et le réseau d'interconnexion.

Dans la machine prototype, chacun des PEs est un microprocesseur 68000, relié par un bus VME à une mémoire locale dynamique de 2 MOctets. Cette mémoire est composée de deux modules de 1 MO, double-port. Ceci permet de recouvrir les accès mémoire de la CPU et de la MSU. En mode MIMD, la mémoire contient à la fois le code exécutable et les données. Le lien vers le réseau d'interconnexion permet à la CPU d'échanger des données avec une autre CPU d'un PE distant. En mode SIMD, la mémoire ne contient que les données; les instructions proviennent directement du MC qui contrôle le PE.

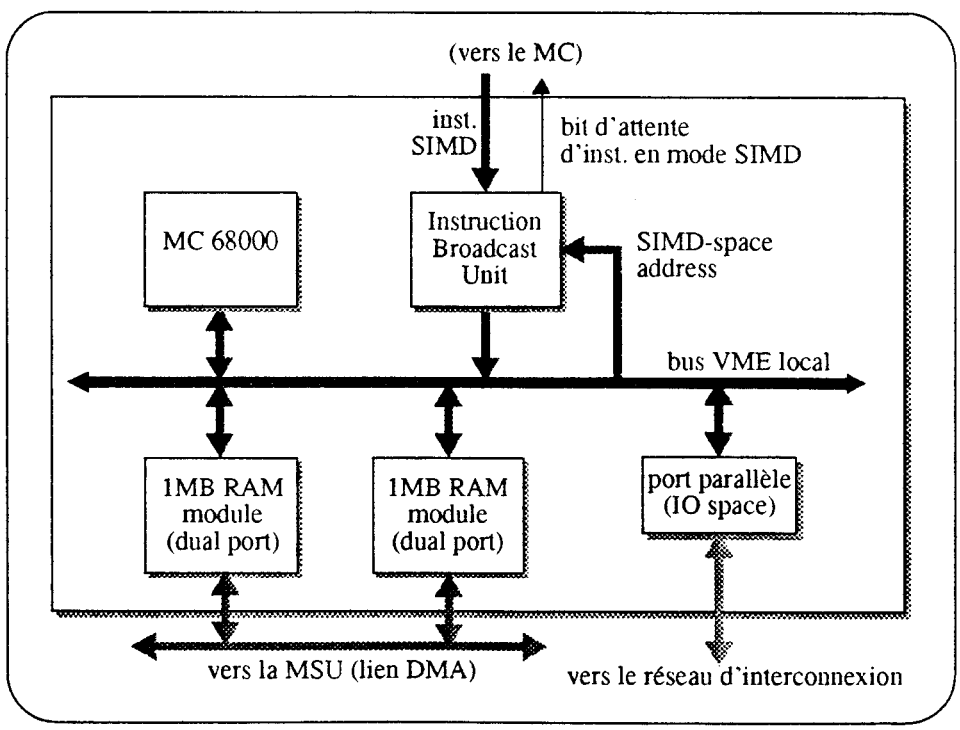


figure 11 synoptique d'un PE de PASM

Le passage en mode SIMD du PE correspond à la lecture d'une instruction dans l'espace d'adressage réservé à ce mode (*SIMD instruction space*). Le MC68000 adresse un espace de 16 MO avec 24 bits d'adresse, qui est découpé en trois parties: 2MO de mémoire locale (code et données), 4MO d'espace mémoire SIMD, et un petit espace d'entrées-sorties.

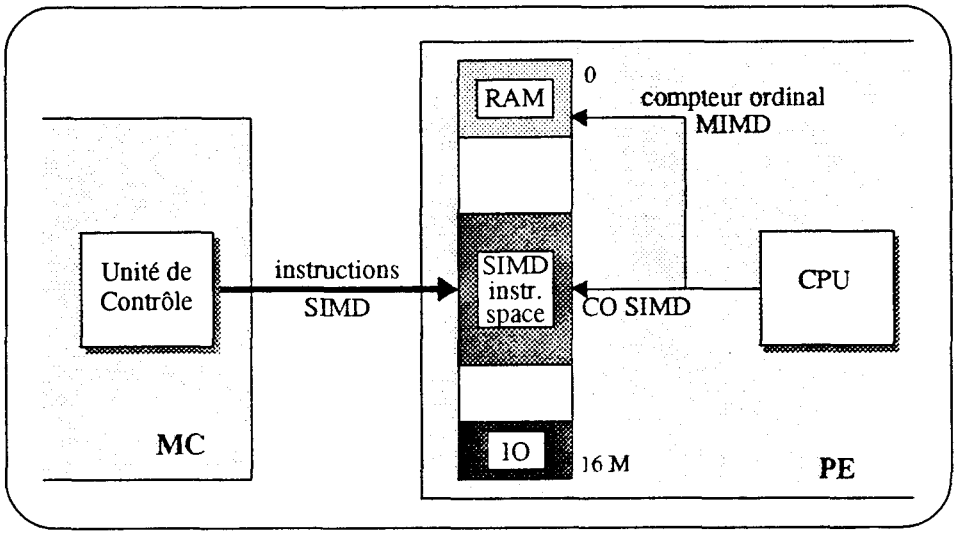


figure 12 espace d'adressage d'un PE

Un accès à l'espace SIMD correspond à une lecture dans l'espace de 4MO réservé. Cet espace n'existe pas; l'adresse est purement logique, et décodée[1] par l'*Instruction Broadcast Unit* (IBU) qui envoie une demande d'instruction au MC dont dépend le PE. A la réception de cette instruction, l'IBU la dépose sur le bus local du PE pour la CPU, comme le fait la mémoire en mode MIMD. Dans les sous-machines comportant plusieurs MCs, la SCU contrôle la synchronisation des PEs en attendant la réception de la demande d'une nouvelle instruction, pour chacun des PEs. Une fois reçues toutes les demandes, la SCU prévient les MCs, qui envoient la même instruction à chacun de leurs PEs. Cette synchronisation s'obtient par un simple arbre de portes ET, de profondeur $\log_2 Q$ [SCHW87]. La perte de temps est relativement faible, de part l'utilisation d'un circuit combinatoire de décision, et la rapidité de la FIFO des instructions SIMD par rapport à la vitesse des RAMs dynamiques locales en mode MIMD.

3.1.3 Le réseau d'interconnexion

Le schéma d'interconnexion des processeurs est souvent le point critique de la réalisation d'une machine parallèle. Il est étroitement lié aux applications envisagées. Le modèle systolique serait la panacée tant attendue, s'il n'était comme nous le verrons trop proche du modèle SIMD. Pour cette raison, [GOTT83] situe l'architecture systolique dans les outils susceptibles de résoudre des problèmes où les flots de contrôle et de données sont réguliers. Certains problèmes résistent cependant à la vectorisation, notamment ceux pour lesquels les décisions dépendent d'un nombre important de données. Proposée par [GOTT83], la machine *NYU Ultracomputer* est donc une machine MIMD à mémoire partagée, pour laquelle le réseau d'interconnexion joue un rôle majeur.

1. Le décodage est effectué sur les 2 bits de poids fort du compteur ordinal de 24 bits. Les 22 bits restants sont ignorés, car le compteur ordinal est géré par le MC en mode SIMD. La CPU de chacun des PEs incrémente donc son compteur ordinal, dont seuls les 2 bits MSB sont significatifs. Lors d'un programme SIMD *long*, i.e. plus de 4MO, le MC peut envoyer aux PEs un "branchement à l'adresse de départ de l'espace SIMD" pour éviter que leur compteur ordinal sortent de cet espace SIMD. Comme l'espace est large, et qu'un branchement de ce type est une action relativement rapide, cela ne devrait pas avoir d'influence réelle sur les performances.

Après avoir écarté les modèles SIMD, systolique et *data flow*, les concepteurs de l'Ultracomputer se sont intéressés à un modèle MIMD. Les données locales et le code exécutable sont situés sur chaque PE (de l'ordre de 4K PEs). Les données partagées se trouvent dans des modules mémoire accessibles par un réseau d'interconnexion de type Oméga, supportant le transfert de messages. Cette interconnexion est plus complexe que le routage de messages par PEs intermédiaires [GOTT83], mais présente pour le réseau Oméga les avantages suivants:

(i) une bande passante linéaire en N, le nombre de PEs,

(ii) une latence en log_2N, la latence est le temps d'accès à la mémoire,

(iii) un réseau comportant $O(N.log_2N)$ éléments identiques,

(iv) des décisions de routage locales,

(v) des accès concurrents à la même mémoire par plusieurs PEs, sans une perte de temps trop importante.

Le réseau d'interconnexion de PASM appartient à une classe très large de réseaux d'interconnexion, rassemblés sous le nom de *réseaux d'interconnexion multiétages*. Il se retrouve notamment dans la machine PASM, mais aussi *la* machine vectorielle française OPSILA [AUGU90] qui est une machine SIMD-SPMD expérimentale (5 Mips et 0,7 MFlops pour 16 PEs...). Ces réseaux de communication sont particulièrement adaptés aux échanges entre PEs et bancs mémoires, lorsque chaque échange peut être traité comme une permutation. "Dans le cas d'une permutation, il n'y a pas de conflits d'accès lorsque les messages progressent d'un commutateur à l'autre dans les étages successifs du réseau", *sic* [ROUX90]. Voyons maintenant plus en détail le réseau utilisé dans PASM.

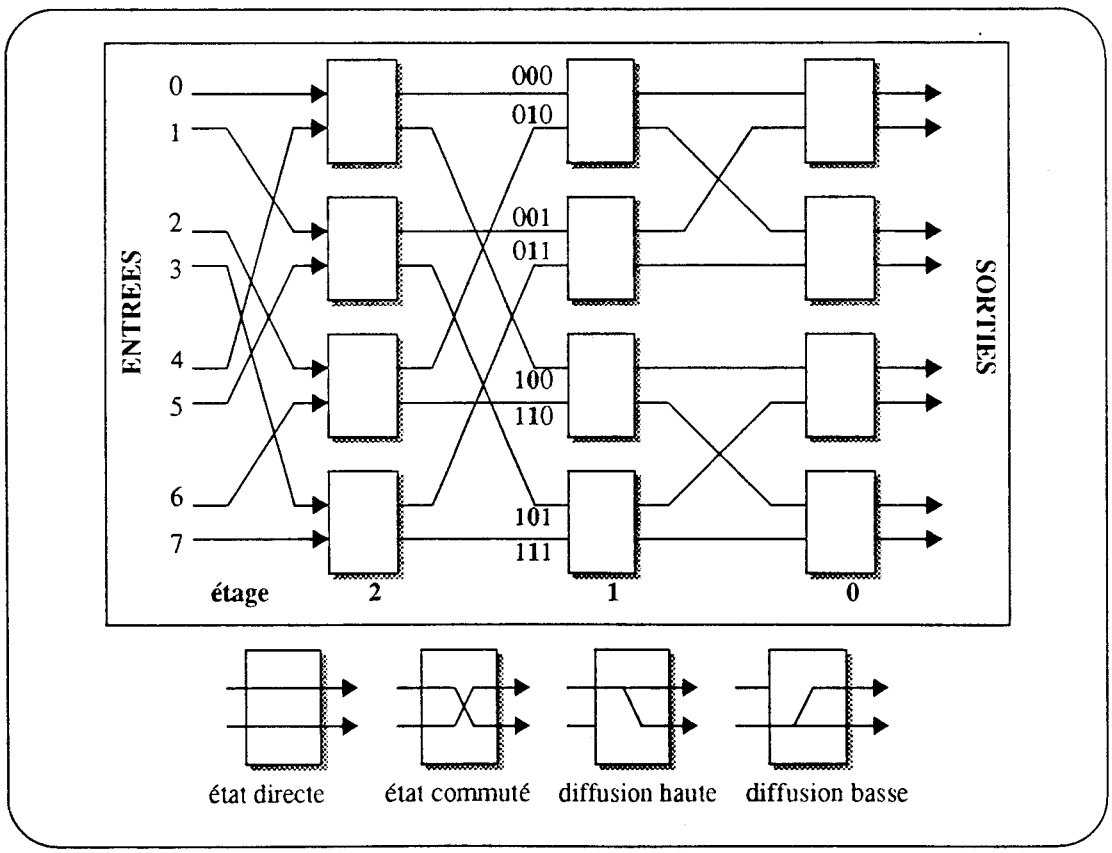
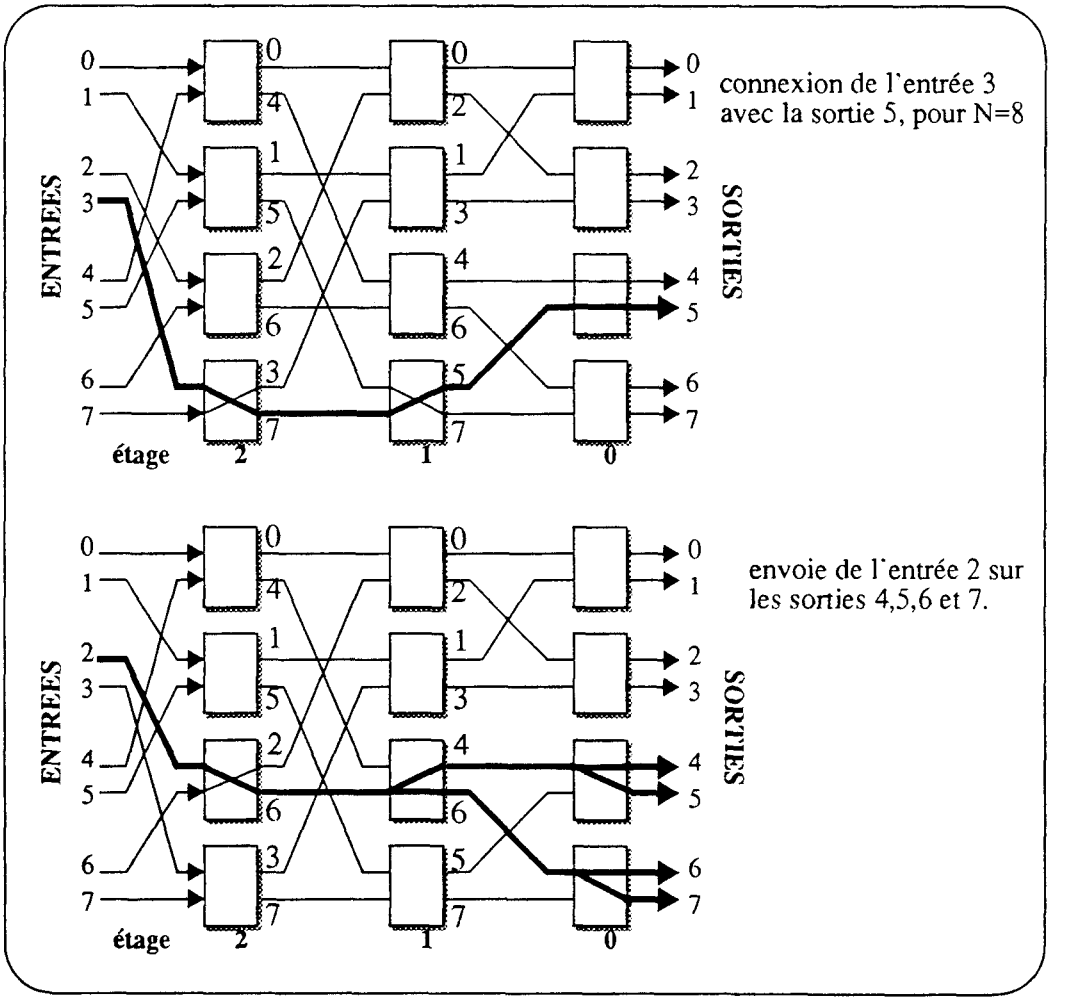


figure 13 le réseau d'interconnexion de PASM, pour N=8

Le réseau multiétages de PASM[1] possède N ports d'entrée, N ports de sortie, et contient $n=log_2N$ étages, de $N/2$ éléments $2x2$ d'échange pour chaque étage. La figure 13 donne le schéma d'un tel réseau pour N=8. Dans ce réseau le $i^{ème}$ PE est connecté sur le $i^{ème}$ port d'entrée et le $i^{ème}$ port de sortie. Les étages sont numérotés de n-1, pour l'étage d'entrée, à 0 pour l'étage de sortie. Le schéma d'entrelacement entre chaque étage est tel que deux liens, dont les numéros ne diffèrent que sur le $j^{ème}$ bit, sont connectés au même commutateur du $j^{ème}$ étage. Chacun des commutateurs prend l'un des quatre états donnés à la figure 13.

L'un des principaux avantages de ce réseau concerne le contrôle du routage des messages, qui se trouve distribué sur chaque commutateur. Pour cela, le message possède une adresse de destination de n bits. Chaque commutateur parcouru examine cette adresse pour déterminer son état. Soit $AD=a_{n-1}....a_1a_0$ l'adresse du port de sortie d'un message, le commutateur du $j^{ème}$ étage qui reçoit celui-ci, le place sur sa sortie haute si $a_j=0$, et sur sa sortie basse si $a_j=1$. Cette méthode de routage présente donc plusieurs avantages: elle est distribuée, l'adresse qui suit le message permet de vérifier que celui-ci arrive à la bonne destination, et cette adresse est simple à générer. De plus cette adresse peut indiquer l'expédition du même message à plusieurs ports de sortie, en ajoutant à l'adresse un masque d'envoie de n bits [SIEG90-1].



1. Anecdote croustillante: ce réseau s'appelle *multistage cube network* dans la description de PASM par [SIEG90-1], et réseau Oméga chez [ROUX90]... Le lecteur, un peu perdu, retrouvera une photographie de famille de ces diverses appellations (non forcément contrôlées) chez [HILL85] aux pages 68-70 de la traduction française. Faisons confiance néanmoins à H.J. Siegel, qui passe pour un spécialiste du domaine... selon [ROUX90].

Un commutateur contient donc un dispositif interne d'arbitrage, qui permet de résoudre les éventuels conflits d'accès. Les deux messages en entrée sont toujours stockés en mémoires d'entrée (une pour l'entrée haute, une pour la basse). Le message y est conservé jusqu'à tant que l'arbitre du commutateur l'autorise à progresser vers la sortie qu'il demande.

Pour terminer, il faut remarquer que ce réseau multiétages est partitionnable, puisqu'il peut être divisé en réseaux indépendants qui conservent tous les propriétés du réseau d'origine [SIEG90-1]. Un réseau cube multiétages de taille N peut être divisé en deux réseaux multiétages de taille N/2. Puisque ces deux réseaux conservent les propriétés du réseau d'origine, ils peuvent à leur tour être divisé en sous-réseaux de taille N/4, N/8...

3.2 Le modèle pyramidal

Les machines pyramidales sont constituées d'étages de machines SIMD, où chacun des processeurs d'un étage communique avec un père de l'étage supérieur, et deux ou quatre fils de l'étage inférieur. Chaque étage est organisé en une machine SIMD, avec une communication aux quatre premiers voisins, comme SPHINX par exemple [MERI89]. Une machine pyramidale est donc une machine MIMD, puisque chaque étage fonctionne indépendamment des étages supérieurs et inférieurs.

Bien qu'il s'agisse de machines novatrices, elles restent tributaires de périphériques rarement parallèles, et, comme pour les machines SIMD, nécessitent des temps de chargement très longs. Ce temps est dans le cas des machines pyramidales d'autant plus important que chaque étage de la pyramide est une petite machine SIMD. Enfin le pyramidal ne semble pas un paradigme, mais un outil bien adapté à certains algorithmes. Jolion fait une synthèse remarquable, à la fois des machines actuelles mais aussi des algorithmes implantés [JOLI90]. Nous reparlerons de l'utilisation d'une pyramide pour la transformée de Hough au chapitre suivant, et nous remarquerons que les processeurs de cette pyramide ont tout intérêt à être puissants.

4 Architectures pour processeurs dédiés

Voici une synthèse des architectures dédiées à la vision, et ne rentrant pas dans les deux catégories précédentes: d'une part les architectures systoliques, notamment la machine WARP qui est une architecture systolique 'programmable', et donc peut être utilisée à différentes fins par réécriture du microcode de cette machine, ainsi que les architectures connexionnistes.

4.1 Architectures sytoliques

L'architecture systolique se présente linéairement: chaque processeur élémentaire est connecté à ses plus proches voisins. Cette disposition rend compte du caractère locale des communications inter-processeurs, et permet dès lors une intégration VLSI plus facile. En effet, si des progrès fantastiques sont réalisés quant au nombre de transistors intégrés sur une puce, le nombre de pattes connectant cette puce à l'extérieur (en l'occurence une carte) est limité et les techniques d'interconnexion de bus (crossbar ou autres) sont chères en composant.

4.1.1 GAPP de NCR

Le GAPP [QUIN89] est un circuit VLSI de 84 broches, qui contient un tableau de 6x12 processeurs élémentaires 1 bit. Chaque PE communique avec ses quatre voisins Nord, Est, Sud et Ouest ainsi qu'avec l'extérieur. Un PE comporte une UAL, une RAM de 128 bits et 4 registres 1 bit. Trois de ces registres sont utilisés par l'UAL; le quatrième sert au communication entrée/sortie sans interférence avec le travail de l'ALU.

Les opérations de base sont réalisées en un cycle. Comme pour toute machine SIMD, l'addition de deux nombres de 8 bits prend 25 cycles; néanmoins, effectuées par les 72 processeurs élémentaires, le GAPP soutient 28 millions d'additions 8 bits par seconde! Ajoutée à la modularité qui permet de créer de vrais grands tableaux de processeurs, ces deux caractéristiques font du GAPP un outil d'emblée séduisant.

Hélas, le GAPP n'a jamais percé comme on aurait pu l'espérer. La raison primordiale provient de la difficulté à programmer un GAPP, et certainement à l'inexistance d'un produit complet soit de simulation soit d'émulation. Un processeur concurrent (quoiqu'intrinsèquement prévu pour le MIMD) lui a semble-t-il 'volé' le marché des applications systoliques[1]: il s'agit bien évidemment du Transputer, aidé en cela par le langage de programmation parallèle Occam.

La remarque du classement de GAPP dans les architectures systoliques, alors qu'il semble plus proche du SIMD, est pertinente. On peut lui objecter l'existence d'une communication indépendante du travail de l'ALU, mais toute micromachine possède cela... Malgrè tout, systolique ou SIMD, il est remarquable qu'un projet ESPRIT existe dont l'objectif est de concevoir un super-composant européen, ELSA pour *European Large Simd Array*, qui intègrerait 128x128 processeurs 1 bit, construit dans le même esprit -si je peux me permettre- que le GAPP. Ce projet trouve d'ailleurs un débouché immédiat dans la réalisation d'une machine de traitement d'images: MASYVE, réalisée au LSIT dans le cadre du PRC-AMN [PRC89]. Ceci indique combien le raté de GAPP n'a pas découragé les concepteurs, loin s'en faut.

1. *Sic transit gloria mundi*, puisqu'il est paraît-il de bon ton de citer ses classiques latins au moment de parler du langage Occam (NDLR cf. [QUIN89] à la page 233).

4.1.2 WARP

Il y trois composants de base dans le système Warp [1]: le tableau de processeurs Warp, l'unité d'interface et le hôte. Les processeurs effectuent les routines de calcul intensif telles que des traitements bas-niveau de l'image. L'unité d'interface gère les entrées-sorties entre le hôte et les processeurs, et génère les adresses et les signaux de contrôle pour le tableau de processeurs. Enfin la machine hôte fournit les données et reçoit les résultats, et peut exécuter les routines qui ne sont pas chargées sur le tableau de processeurs (routines de décision, par exemple). La figure 14 décrit l'architecture d'un système Warp.

Le tableau de processeurs se présente comme une chaîne de cellules identiques. Les données transitent à travers cette chaîne de processeurs par les deux canaux de communication (X et Y). La direction du cnanl Y est statiquement configurable. Ceci présente un double avantage, pour les algorithmes nécessitant des informations évaluées par la dernière cellule et renvoyées en arrière, mais aussi pour des algorithmes requérant des échanges entre cellules adjacentes. Les adresses générées par l'unité d'interface circulent linéairement dans le réseau par l'intermédiaire du canal d'adresse. Elles contrôlent ainsi le fonctionnement des cellules.

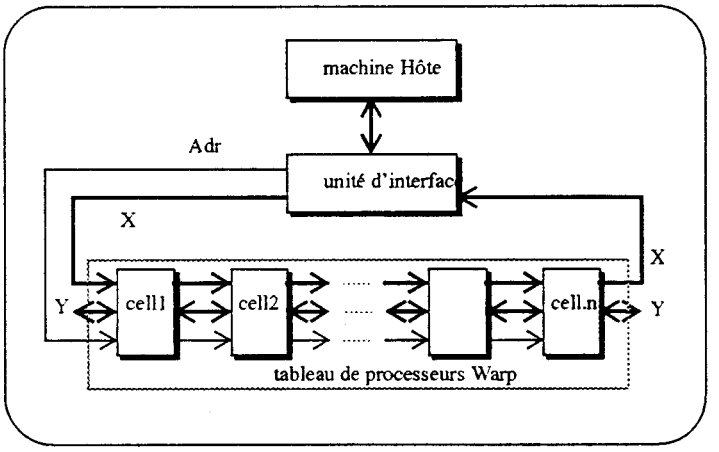


figure 14 Architecture du système Warp

Chaque cellule Warp est une micromachine à microcode horizontal, avec son propre séquenceur et sa mémoire de microprogramme de 8K microinstructions. Le chemin de données est constitué d'un multiplieur (Mpy) de 32 bits flottant, d'un additionneur flottant 32 bits (Add), de deux bancs mémoire distincts pour les variables locales résidantes et temporaires (Mem), une 'queue' pour chacun des canaux de communication (XQ, YQ et AdrQ), et une ensemble de registres de travail pour les deux unités flottantes (AReg et MReg).

Toutes ses composantes sont connectées entre elles par un crossbar. Les adresses peuvent être calculées localement par le séquenceur (Agu pour Address Generation Unit) ou récupérées directement sur le bus d'adresse à travers une 'queue' d'adresses (AdrQ) comme dans toute micromachine.

La 'queue' est un moyen de stockage par lequel toute écriture se fait en fin de liste, tandis qu'une lecture peut se faire à n'importe quelle position dans la liste. Les queues sont automatiquement compactés à chaque retrait de données.

L'architecture systolique permet de résoudre le goulet d'étranglement propre au architecture Von Neumann, c'est à dire une limitation de la capacité de calcul dûe à la largeur de bande passante des mémoires. A ce sujet, [KUNG82] remarque: "*Systolic architectures, which ensure multiple computations per memory access, can speed up compute-bound computations without increasing I/O requirements*". Cependant, s'il n'y a plus de problème d'accès aux objets externes, il n'en reste pas moins qu'en interne les demandes sont toujours proportionnelles au nombre d'opérations effectuées, non négligeable de part la présence de deux unités de calcul rapides. Aussi un crossbar permet d'interconnecter 6 entrées de données à 8 sorties (dont quatre vers les unités fonctionnelles). Bien que beaucoup plus coûteux en terme de portes, le crossbar rend plus facile l'écriture d'un compilateur, comparé à l'utilisation de bus partagés où des situations de blocages sont toujours possibles.

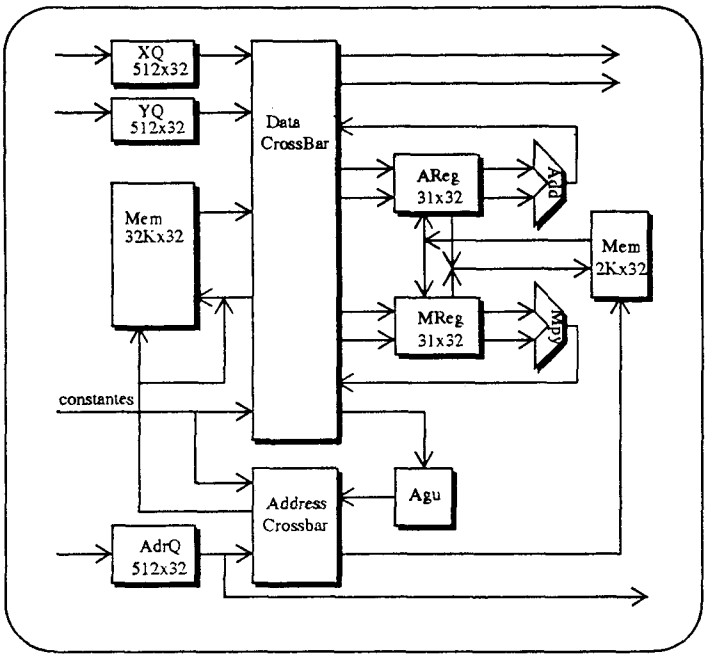


figure 15 Architecture d'une cellule Warp

La cellule Warp peut paraître dotée d'une mémoire trop importante pour une application systolique mono-dimensionnelle; en fait ce choix a été spécialement retenu lors de la conception du processeur Warp pour permettre la simulation d'une structure bi-dimensionnelle avec un réseau linéaire de processeurs. On multiplexe alors temporellement le fonctionnement de chaque cellule pour simuler toute topologie d'interconnexion autre que linéaire.

L'autre avantage de ce processeur est relatif à sa puissance de calcul, qui lui permet de modéliser soit une architecture parallèle grain-fin (PE peu puisssant, ex. CM), soit une architecture parallèle gros-grain (PE très puissant, ex. Hypercube). Dans le cas d'une demande forte sur les entrées-sorties (cas le plus souvent du systolique), la capacité de transfert IO de chaque cellule est suffisante pour soutenir un débit de communication important entre cellules voisines. Dans le cas contraire d'un calcul intensif, la mémoire de microprogramme (8K) et les deux unités flottantes rapides (32bits, flottant) permettent de configurer facilement une machine à parallélisme gros-grain.

Cette configuration peut alors s'attaquer à des opérations plus globales, contrairement à une architecture systolique dont chaque sortie dépend d'un petit nombre d'entrées. Nous reviendrons au chapitre suivant sur l'utilisation d'une machine à base de processeurs Warp, pour une application de traitement d'images.

4.1.3 Architectures systoliques pour la convolution

Les réseaux de convolution semblent un bon outil d'introduction à l'architecture systolique [QUIN89]. De plus, les convolutions sont utilisées en traitement d'images, comme nous le montrerons au prochain chapitre. Voyons d'abord ce qu'est un filtre.

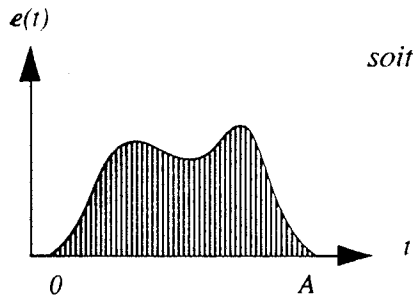
Définition: On appelle filtre F toute transformation linéaire, invariante dans le temps. Soit $x(t)$ un signal évoluant au cours du temps, la réponse du filtre F est la fonction $y(t)=F(x(t))$.

F présente donc les propriétés suivantes:

$$F(\lambda x_1 + \mu x_2) = \lambda F(x_1) + \mu F(x_2)$$

$$si \quad y(t) = F(x(t)) \quad alors \quad y(t+\tau) = F(x(t+\tau))$$

On appelle réponse *percussionnelle* ou *impulsionnelle* l'image de $\delta(0)$ par F, où $\delta(0)$ est la fonction de Dirac en $t=0$. Soit $h(t)$ cette réponse impulsionnelle associée au filtre F, pour une entrée $e(t)$ quelconque du filtre F, cette entrée peut être découpée sur l'intervale [0,A] en n tranches, de la manière suivante:



soit la suite $$e_n(t) = \sum_{k=0}^{n-1} \frac{A}{n} \cdot e\left(\frac{k \cdot A}{n}\right) \cdot \delta\left(\frac{k \cdot A}{n}\right)$$

la réponse de F à la suite $e_n(t)$ *est:*

$$F(e_n(t)) = \sum_{k=0}^{n-1} \frac{A}{n} \cdot e\left(\frac{k \cdot A}{n}\right) \cdot h\left(t - \frac{k \cdot A}{n}\right)$$

on montre que $$\lim_{n \to \infty} (e_n(t)) = e(t)$$

et, en passant aux limites dans les réponses,

$$F(e_n(t)) \to \int_0^A e(u)\, h(t-u)\, du = h(t) \otimes e(t) \qquad pour \quad n \to \infty$$

La réponse à une entrée $e(t)$ est le produit de convolution de $e(t)$ avec la réponse impulsionnelle $h(t)$. Les convolutions non récursives correspondent à des filtrages à réponse impulsionnelle finie. Soit la donnée d'une suite $x_1, x_2, \ldots$ de données, alors pour tout entier $i \geq k$,

$$y_i = a_1 \times x_i + a_2 \times x_{i-1} + \ldots + a_k \times x_{i-k+1}$$

est la réponse du filtre F auquel est associé l'ensemble des poids $a_1, a_2, \ldots a_k$.

La solution de la figure 16 est très répandue en filtrage numérique. A chaque top d'horloge, une nouvelle donnée apparaît en entrée. Une fois les registres mis à jour, les multiplications par les coefficients du filtre ont lieu en parallèle; le résultat final est évalué par une addition sur les k valeurs obtenues

par multiplication. Le coût de cette addition finale, tant matériel que temporel, amène la synthèse d'une architecture systolique adaptée à ce calcul [QUIN89].

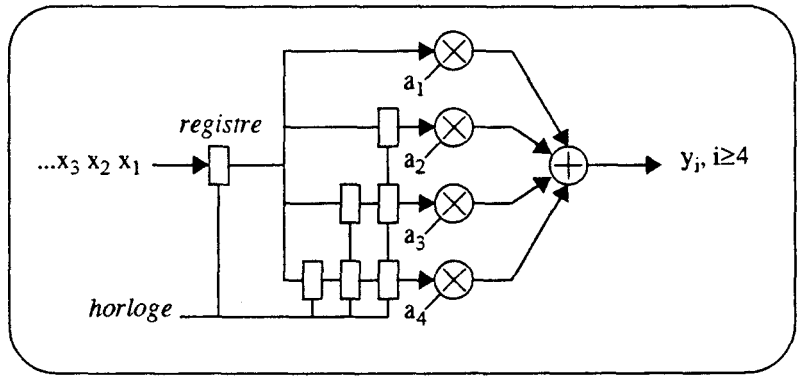
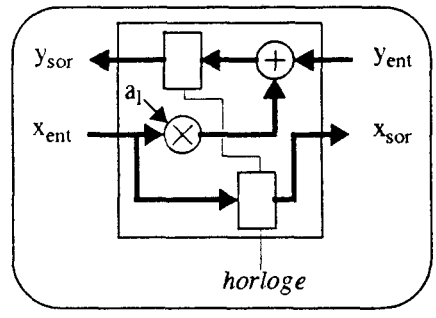


figure 16 solution architecturale à la convolution (k=4)

L'idée est de calculer localement la somme partielle, en faisant circuler en sens inverse les x_i d'une part, les y_i d'autre part. Pour cela, on utilise la cellule suivante:



qui à chaque top d'horloge modifie y_{sor}, $y_{sor}=y_{ent}+a_1.x_{ent}$, et x_{sor}, $x_{sor}=x_{ent}$. Il y adonc deux flots de données dans le réseau, ce qui implique une distribution des nouveaux x_i tous les deux tops d'horloge. Ceci est général [QUIN89]; dans le cas où deux flots de données circulent en sens inverse, il est nécessaire d'ajouter un temps d'attente entre deux variables délivrées pour que celles-ci rencontrent toutes les données circulant en sens inverse, et non la moitié.

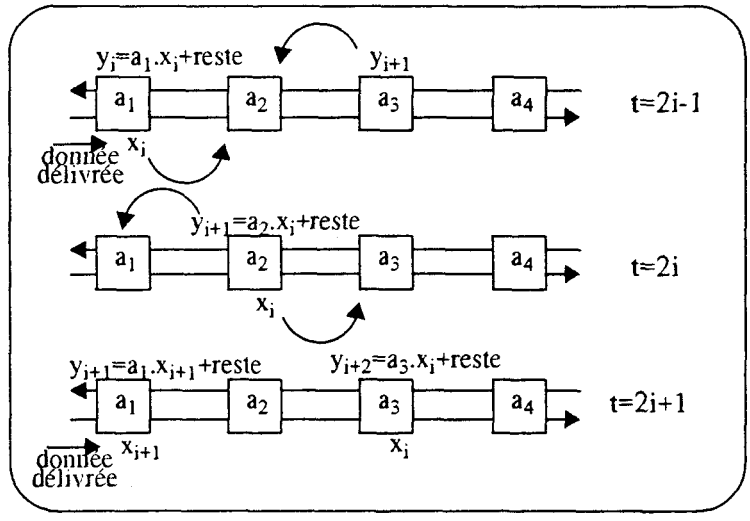


figure 17 déroulement d'une double circulation

[QUIN89] propose enfin d'appliquer le vieil adage: *diviser pour régner*, afin d'accroître encore les performances du circuit, c'est à dire d'envoyer des valeurs tous les tops d'horloge. L'idée consiste à diviser un problème en n sous-problèmes, facilement résolus et fusionnés ensuite. Pour notre circuit, cette approche revient à séparer en deux parties distinctes le calcul de convolution:

$$y_i = a_1 \times x_i + a_2 \times x_{i-1} + \ldots + a_k \times x_{i-k+1}$$

$$\textit{devient} \quad \begin{cases} y^1{}_i = a_1 \times x_i + a_3 \times x_{i-2} + a_5 \times x_{i-4} + \ldots \\ y^2{}_i = a_2 \times x_{i-1} + a_4 \times x_{i-3} + a_6 \times x_{i-5} + \ldots \end{cases}$$

On construit alors deux réseaux, un pour le calcul des y_i^1 et un autre pour les y_i^2. Une cellule de sortie combine les deux résultats pour obtenir $y_i=y_i^1+y_i^2$. Dans ce nouveau réseau, une nouvelle variable x_i est délivrée à chaque top d'horloge.

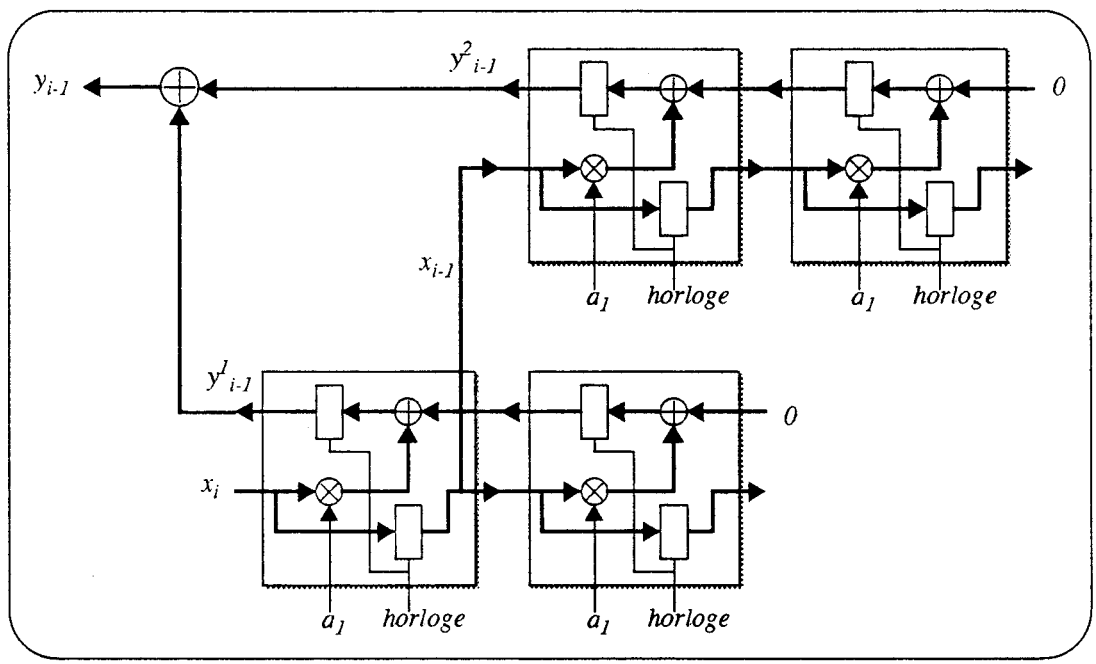


figure 18 réseau de convolution efficace tous les tops d'horloge

4.2 Architectures pour le connexionnisme

Nous ne mentionnons ici que les problèmes rencontrés par les architectures connexionnistes. Parmi celles-ci, les architectures SIMD sont les plus adaptées aux réseaux d'automates cellulaires à topologie de connexion 4-voisins. La Connection Machine s'avère efficace pour les réseaux d'automates cellulaires à topologie plus complexe. Le modèle le plus employé dans le connexionnisme est celui à couches, avec la rétro-propagation du gradient comme algorithme d'apprentissage. [GASC90] propose un circuit VLSI d'émulation de ce modèle, possédant un apprentissage câblé en interne.

La difficulté est de trancher entre un calcul analogique ou numérique. L'inconvénient de l'analogique tient dans la programmation limitée qu'offre le circuit, et la limite du codage des niveaux des synapses. Son avantage est la rapidité et la compacité des circuits obtenus. Le laboratoire d'électronique de Lausanne a développé un circuit analogique de 2 neurones et 128 synapses.

A l'opposé les circuits numériques se trouvent face à un choix: travailler en nombres flottants ou limiter en nombres fixes la précision des calculs. Dans tous les cas, la connexité des cellules reste le critère qui limite chaque réalisation. On ne peut, à la manière du cerveau humain, connecté 10000 synapses sur chaque neurone dans un boîtier 2D. [GASC90] propose à ce sujet une implantation systolique du calcul des coefficients. Les coefficients synaptiques sont stockés dans un registre à décalage propre à chaque neurone: les coefficients d'une ligne de la matrice de connexions entre couches. Lors du calcul, ces coefficients circulent et sont multipliés par l'état du neurone auquel ils sont liés pour donner à la fin l'état du neurone de la couche suivante. Réalisé en ASIC *full custom*, les performances attendues sont de l'ordre de 10 ms pour retrouver une forme apprise.

5 Machines à communications *via* un bus

La technique des circuits spécifiques s'améliorant, il est vraisemblable que de plus en plus de traitements seront *câblés* plutôt qu'effectués sur des processeurs à usage général. Cette remarque anodine risque de ne pas l'être pour ceux-là même qui sont convaincus que ces processeurs généraux fonctionneront toujours plus vite. Nous argumentons à l'encontre de cette supposition que *(i)* cette croissance des performances atteindra (certainement bientôt) ses propres limites physiques, au moins pour la technologie silicium, *(ii)* et que les performances actuelles sont bien en-deçà des besoins réels, et enfin *(iii)* que l'antropomorphisme appliqué à la vision participe à une certaine conception de la recherche informatique.

Ceci dit, si le parallélisme massif ouvre de nouvelles perspectives pour la construction de machines fonctionnant effectivement en temps réel, il se conçoit de deux manières: homogène ou hétérogène. L'architecture SIMD est le plus bel exemple du parallélisme homogène, mais qui se réalise par une machine coûteuse et relativement figée. Le concept de processeurs dédiés est la base des machines hétérogènes, où chaque processeur a son propre mode de fonctionnement: quels accès mémoire? quelles synchronisations?, et avec quels autres processeurs?... Les mécanismes de communications inter-processeurs pèsent néanmoins sur le budget de telles machines, qui pour des critères de flexibilité et de *ratio* performance/coût, ont adopté la structure de bus comme support de communications.

Dans ce paragraphe, nous retraçons l'émergence de ce type d'architecture si particulier au domaine du traitement d'images (ou du signal), notamment en présentant la machine Tospics. Ensuite nous parlerons des calculateurs puissants, beaucoup utilisés en robotique ou dans les systèmes embarqués. L'architecture de bus en anneau confère à ces machines des performances excellentes. Enfin nous présenterons les machines d'évaluation, qui sont conçues comme des ateliers *matériel-logiciel* pour la mise au point d'applications flexibles en vision.

5.1 TOSPICS

Cette machine nous paraît être une bonne référence pour aborder les "machines orientées bus", terme employé par [DANI81]. L'arrivée de circuits dédiés et leur mise en parallèle offrent aux concepteurs de systèmes de vision un champs d'applications élargies, avec des coûts matériels en baisse et une plus grande souplesse de programmation. Tospics [KIDO83] fait partie de ces machines développées par les universités et entreprises japonaises autour d'idées simples: processeurs dédiés, microprogrammation, mémoire d'images, bus pour le partage des données.

Cette machine comporte donc une mémoire de quatre images de 512x512 pixels sur 8 bits, avec la possibilité d'accroître cette taille mémoire suivant les applications envisagées. Une mémoire de données rapide permet de stocker quatre lignes pour les accès consécutifs à des voisinages 4x4 de l'image (gradient, convolution,...). Quatre processeurs ont été spécialement dessinés pour être placés en parallèle sur le bus: un circuit de convolution 4x4; un autre pour la recherche d'une valeur maximale, minimale ou moyenne dans un voisinage 3x3; un troisième pour les opérations logiques à partir d'un masque 3x3 (transformations morphologiques); enfin un quatrième circuit d'étiquetage de régions, qui permet de détecter celles-ci.

Au niveau du contrôleur de la mémoire d'images, on trouve deux générateurs d'adresses qui calculent en parallèle les adresses des pixels allant vers, et venant de, la mémoire. De plus une ALU permet de calculer lors de ces transferts les opérations affines. Le cycle de base sur un pixel prend 1 µs. Les opérations de base, telles que la convolution, demandent donc 0,3s sur Tospics.

Encore machines de chercheurs à cette époque (au début des années 1980), celles-ci ont évolué vers les applications des mondes militaires, médicaux, robotiques... Elles ont participé à la naissance des calculateurs dédiés aux traitements du signal et de l'image, en amenant sur le marché des architectures rapides pour la mise en oeuvre des techniques reconnues et usitées. Dorénavant les concepteurs de systèmes spécialisés peuvent y faire appel, et trouvent dans ces machines la puissance et la spécificité que requièrent leurs applications.

5.2 Calculateurs puissants

Ces machines sont généralement réservées aux traitements de bas niveau, mais aussi de niveau intermédiaire tel que l'approximation polygonale. Ainsi le calculateur CAPITAN [GAIL84-1] sert de machine de base pour le dévelopement d'une machine de vision, pour le calcul visuel des distances et du mouvement [FAUG88]. L'orientation temps réel des applications est le critère de base pour recourir à ces calculateurs coûteux, qui fournissent souvent un support au développement d'une machine plus puissante. Construits autour d'un bus VME, on peut envisager de connecter des processeurs conçus par l'utilisateur et intégrés au système grâce au bus.

Cependant, si des contraintes de rapidité de communications l'exigent, la structure du bus peut être dérivée pour soutenir des débits importants. En effet, les calculateurs sont généralement des machines de type MIMD, construites autour de processeurs commercialisés très puissants. Avec des horloges à 20ns, le fonctionnement et la communication de ces processeurs deviennent difficiles. Les bus classiques ne suffisent plus à alimenter en données les processeurs, car ils sont limités en vitesse du fait de leur architecture globale. On en vient donc au concept du *bus en anneau*, implanté dans le calculateur CAPITAN.

5.2.1 CAPITAN

Le CAlculateur Parallèle pour l'Imagerie, le Traitement du signal et l'Analyse Numérique a été développé comme structure d'accueil de processeurs VHSIC (Very High Speed Integrated Circuit) conçus spécialement pour des tâches courtes, relativement indépendantes, et donc exécutées en parallèle, et pipelinées entre elles [GAIL84-2]. Pour cette raison, les transferts s'effectuent par blocs de données, avec des adresses soit consécutives soit aléatoires, mais toutes définies à l'initialisation du transfert: le processeur connaît toutes les données dont il a besoin. Aucune adresse d'éléments du bloc ne doit être calculée en fonction des premières données accédées dans ce bloc.

Ainsi les contraintes sur le mécanisme d'échange de Capitan sont réduites [GAIL84-2], et se résument à:

(i) la résolution des conflits d'accès inévitables dans les applications parallèles de ce type,

(ii) un débit d'échange élevé,

(iii) et peu de contraintes sur le délai d'acheminent: l'utilisation de mémoires tampon diminue les risques d'absence de données à l'initialisation d'une tâche élémentaire.

L'architecture de Capitan relève du modèle MIMD, où chaque processeur possède sa propre mémoire locale de donnée (cf. figure 19). Le partage des données globales, par exemple les images, est supporté par deux bus en anneau qui peuvent se diviser par logiciel entre processeurs voisins. Cette division des bus offre la possibilité de configurer des blocs de processeurs fonctionnant en parallèle, tout en

pipelinant ces blocs entre eux: des deux bus en anneau, l'un est l'entrée du bloc et l'autre la sortie. Nous renvoyons les lecteurs intéressés par cette reconfigurabilité de la machine à [GAIL84-1]. Le débit des bus sur une telle architecture est critique; le bus en anneau est alors retenu car intrinsèquement plus rapide que le bus trois états classique.

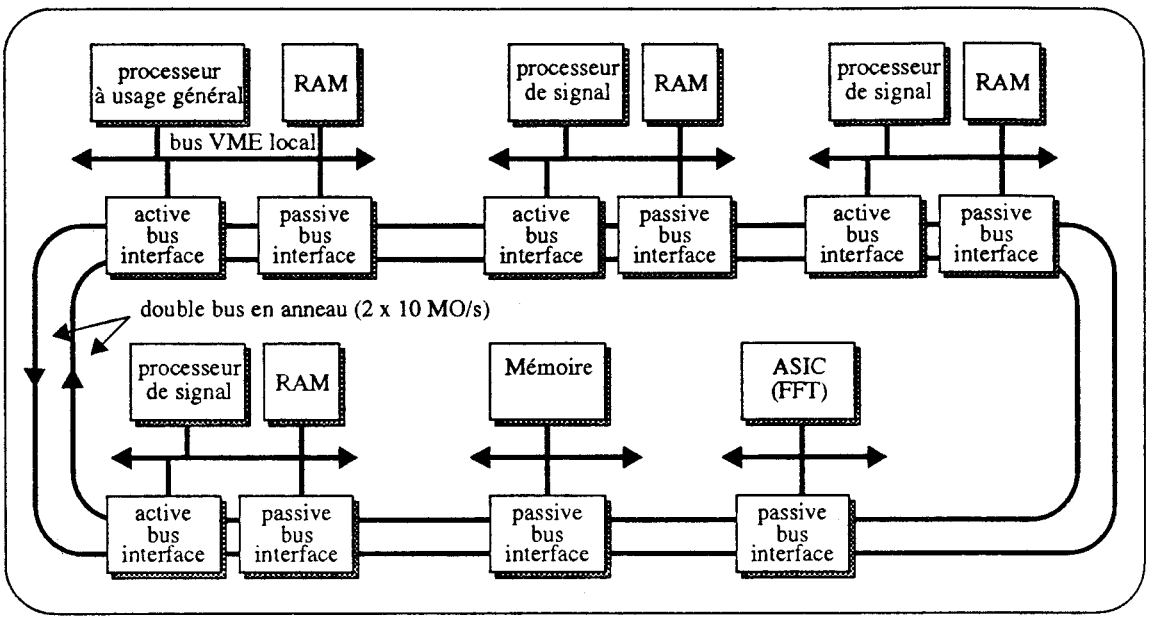


figure 19 Architecture générale du calculateur CAPITAN

5.2.2 *Le bus en anneau*

La flexibilité d'un bus trois états classique masque généralement la faiblesse de son débit. Cependant certaines applications demandent à la fois cette flexibilité et un important débit, comme par exemple les calculateurs orientés temps réel. Le bus en anneau est alors une bonne solution, peu onéreuse comparée aux mécanismes d'interconnexion des machines MIMD (faire une référence).

Une telle structure est comparable à un long registre à décalage rebouclé sur lui-même. Chaque processeur[1] connecté à ce bus, lecteur ou/et rédacteur, possède devant lui un multiplexeur suivi d'un registre. A chaque cycle de l'horloge, les données avancent d'un registre et chacun des processeurs peut lire une donnée ou/et en placer une nouvelle sur le bus. Cette architecture de bus se retrouve souvent dans les calculateurs.

Lorsqu'un processeur veut envoyer un message, il attend le passage d'une case vide et y dépose ce message. Contrairement au bus tois états, le message déposé arrive au processeur destinataire en au plus n cycles d'horloge pour une configuration à n processeurs. Quant au *débit* de ce bus, il est identique que celui d'un bus trois états pour une même fréquence d'horloge.

Or l'avantage de cette structure tient dans la vitesse de l'horloge de ce bus, vitesse liée à des considérations électriques. En effet, puisque les lignes entre processeurs voisins sont beaucoup plus courtes qu'une ligne continue depuis le premier jusqu'au dernier des processeurs, et qu'en plus il n'est pas fait usage de mécanismes trois états qui ralentissent toujours au moins l'écriture, le bus en anneau fonctionne

1. Le terme *processeur* est employé dans son acception la plus générale, puisqu'il peut s'agir de processeurs à usage général, de DSP, voire d'ASICs dédiés à la réalisation d'un certain type de traitement, et aussi de mémoires, d'interfaces d'entrée-sortie ou de capteurs. Ces éléments peuvent alors être maîtres ou esclaves de ce bus.

à une fréquence plus élevée que les bus trois états (le rapport est de 2 à 3).

Ainsi l'horloge du bus en anneau de CAPITAN fonctionne à 50ns pour une configuration à 16 processeurs en technologie TTL[1] [GAIL84-1]. D'ailleurs Gaillat remarque fort justement que la réalisation d'un bus trois états pour 16 processeurs à 200ns demanderait beaucoup de soins, et nécessiterait des circuits de contrôle dissipant beaucoup de puissance. Les structures actuelles de bus en anneau fonctionnent à des vitesses de l'ordre de 33Mhz, en VLSI CMOS ! Cette technique permettra donc d'atteindre des débits de l'ordre du Gbits/s au cours de la décennie. Ces vitesses seront alors compatibles aux besoins de certains processus de traitements d'images et du signal.

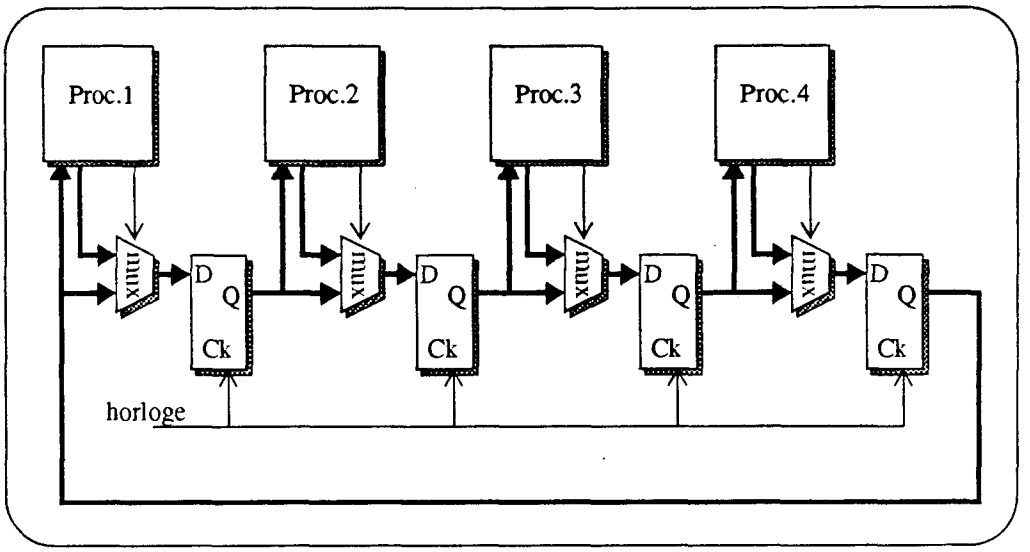


figure 20 Bus en anneau pour 4 processeurs

5.3 Les machines d'évaluation

5.3.1 La machine PRIVE

Le Processeur d'Images Vidéo d'Evaluation, PRIVE 2 [DERU85], consiste en une machine construite à l'aide de plusieurs processeurs de traitement spécialisés, qui extraient chacun une certaine information et fonctionnent au rythme d'échantillonage du capteur d'images (cf. figure 21). La mise en oeuvre des différentes unités de traitement est programmable. Il existe une bibliothèque de modules orientés temps réel, qui sont effectués sur des processeurs câblés [DERU84] ou des processeurs à usage général. Le flot de données est le plus souvent *pipeliné*, pour répondre aux demandes importantes des processeurs en transfert de données.

Un bus indépendant permet une communication inter-processeurs, qui facilite le choix des données pertinentes à la phase de reconnaissance. Le système devient complexe, mais se rapproche de la structure hétérarchique des systèmes de vision des êtres vivants. Contrairement aux calculateurs, ce bus n'est pas particulièrement rapide; le plus souvent, il s'agit d'un bus VME ou Multibus. Ces machines sont construites pour évaluer des traitements câblés originaux, provenant d'universitaires, ou analyser des

1. Les performances annoncés par [GAIL84] datent un peu. Néanmoins l'écart s'est conservé, et participe au succès du bus en aneeau pour toutes les architectures de calculateurs parallèles. L'apparition de processeurs puissants et rapides rend encore plus critique les communications entre eux; c'est le cas pour les applications militaires notamment...

images dans un contexte peu contraint: médical, vision industrielle,... Le temps gagné par les processeurs spécialisés suffit amplement aux besoins des applications.

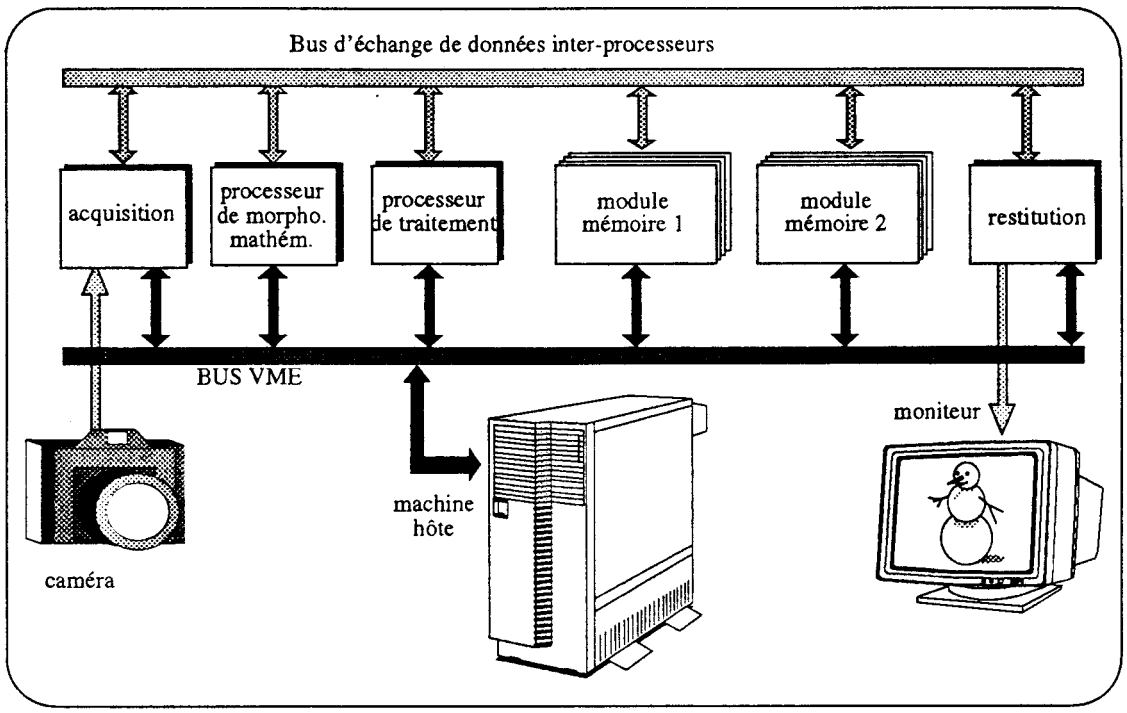
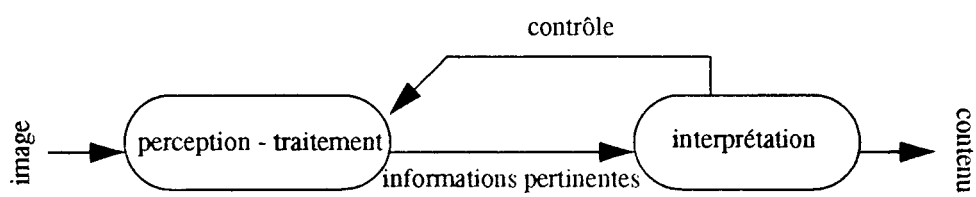


figure 21 schéma synoptique de PRIVE 2

Extrapolant cette construction, [MAFF86] propose deux flots de données dans une machine de vision: un flot de données habituel (sic) de la partie traitement vers la partie interprétation, mais conjoint à un flot inverse qui permette de finaliser les traitements de bas niveau par un contrôle de l'unité d'interprétation. Le projet CODEVI, COprocesseur DE VIsion, s'il tente de s'inspirer de ce modèle [MAFF86], n'aboutit qu'à une structure somme toute banale de machine de vision, telle qu'on peut la retrouver dans des produits commerciaux.



Les cartes de la série 150-151 d'Imaging Technology Inc. ainsi que celles de la société Matrox, MVP et Image Series, méritent d'être cités, puisqu'elles proposent une architecture orientée bus: chaque carte est disposée sur un bus, relié à une station de travail. D'autres produits existent bien entendu... A l'aide de ce matériel, l'utilisateur crée son système en fonction de ses besoins. Chacun a donc la possibilité de construire son système de vision "à la carte". Fait de société, les constructeurs proposent des cartes mères acceptant des modules (processeurs dédiés) installés au choix.

Ces produits proposent des cartes mémoire: *frame buffer* pour mémoire de trame, des cartes CPU, où l'ajout d'un voire plusieurs processeurs RISC amène la puissance nécessaire aux applications recherchées. Les processeurs câblés proposés appartiennent généralement à la famille des traitements de bas niveau: convolution, filtrage, analyse morphologique,...; toutes opérations pour lesquelles on possède

des implantations matérielles efficaces et bien connues, et qui ont été intégrées à l'aide de circuits spécifiques ASIC. Les langages de programmation sont soit le Fortran soit le C, pour les plus répandus. Les constructeurs offrent des cartes accélératrices avec des "*packages*" logiciels compatibles avec leur offre standard: par exemple les toutes nouvelles cartes VX et MVX de chez SUN, qui permettent d'accélerer les programmes utilisant SUNVIEW, entre autres. Le marché est important pour ce genre de produits, mais difficile car encore trop spécialisé.

6 Conclusion

Pour conclure, cette étude quelque peu succincte nous a permis de constater l'existence d'un lien étroit entre les algorithmes et les architectures qui les effectuent. Le lecteur aura pu se rendre compte combien la parallélisation, qui peut sembler *a priori* la solution à l'accélération de ces algorithmes, implique une évaluation précise des objectifs et moyens. Objectifs, parce que tout algorithme n'est pas forcément parallélisable, et qu'une bonne parallélisation n'est pas automatiquement la plus naturelle. Moyens, parce que certaines architectures offrent un gain satisfaisant par la parallélisation, tandis que d'autres ne permettent pas de tirer partie pleinement du *ratio* coût/performance.

Appliquées à la vision, l'efficacité des différentes architectures dépend étroitement des algorithmes qu'elles réalisent. Il n'y a donc pas ici de paradigme, et on ne peut envisager aujourd'hui de dire que la machine de vision du futur aura tel type d'architecture. Ce constat qui n'aboutit à aucun choix est en fait porteur d'information. En effet, si aucune architecture peut à elle seule prétendre à l'universalité, par contre il est évident que chacune est apte à résoudre au mieux une classe d'algorithmes donnée. Aussi, depuis les possibilités offertes par les circuits intégrés spécifiques, les ASICs, la conception d'une machine de vision à modules architecturaux dédiés peut s'envisager.

Références Bibliographiques du Chapitre 1

[AUGU90] Auguin M, Boéir F, Dalban JP,
"Synthèse et évaluation du projet OPSILA",
TSI, vol. 9, No 2, 1990

[BALL82] Ballard DH, Brown CM,
Computer Vision,
Prentice-Hall inc., 1982

[BARN68] Barnes G.H., & al,
"The ILLIAC IV Computer",
IEEE Trans. on Computers, Vol. C-17, No 8, August 1968

[BATC80] Batcher K.E.,
"Design of a Massively Parallel Processor",
IEEE Trans. on Computers, Vol. C-29, No. 9, Sept 1980

[DANI81] Danielsson PE, Levialdi S,
"Computer architectures for pictorial information systems",
Computer, pp.53-67, Novembre 1981

[DERU84] Derutin JP, Alizon J, Gallice J,
"Determination de la position d'un objet manufacturé en temps réel vidéo par processeurs cablés",
1er Colloque Image, CESTA, Mai 1984

[DERU85] Derutin JP, Alizon J, Bonton P et Gallice J,
"Processeur d'Image Vidéo d'Evaluation: PRIVE 2",
10ème colloque sur le traitement du signal et ses applications, Nice, vol2, Mai 1985

[FAUG88] Faugeras O,
"Les machines de vision",
La Recherche, vol. 19, pp. 1334-1346, Nov. 1988

[GAIL84-1] Gaillat G,
"Le calculateur parallèle CAPITAN: 600 Mips pour l'imagerie temps réel",
1er colloque Image, vol. 1, pp.473-481, Mai 1984

[GAIL84-2] Gaillat G,
"CAPITAN: un calculateur parallèle pour le TI embarqué à bord de sattelites",
4ème Congrès AFCET sur la RF et l'IA, PARIS, pp.567-586, Janvier 1984

[GASC90] Gascuel JD, et al,
"Implementation of internal annealing in a VLSI feedback neural network chip",
Intern. workshop on Algorithms and Parallel VLSI Architectures, Pont-à-Mousson, Juin 90

[GOTT83] Gottlieb A, et al,
"The NYU Ultracomputer-Designing an MIMD Shared Memory Parallel Computer",
IEEE trans. on Computers, Vol C-32, No 2, February 1983

[HILL85] Hillis D,
La machine à connexions,
traduction française, MASSON, 1988

[HORD82] Hord R.M.,
"The ILLIAC IV, the first supercomputer",
Computer Science Press, Rockville, Md, 1982

[JOLI90] Jolion JM,,
"Analyse d'images: le modèle pyramidal",
Traitement du Signal, vol. 7, No 1, 1990

[KIRK83] Kirkpatrick CD, et al,
"Optimization by simulated annealing",
Science, vol 220, pp.671-680, May 1983

[KUCK68] Kuck DJ,
"Illiac IV software and application programming",
IEEE Trans. on Comp., vol C-17, No 8, pp.758-770, Août 1968

[KUNG82] Kung H.T.,
"Why Systolic Architectures?",
Computer, vol.15, Jan.1982, pp.37-46

[MAFF86] Maffres M,
CODEVI une architecture adaptée à la vision par ordinateur,
Thèse Docteur Ingénieur, INPG (86-INPG-0056), Juin 1986

[MERI89] Mérigot A, et al,
SPHINX, un processeur pyramidal massivement parallèle pour la vision artificielle,
AFCET, RFIA, tome 1, Paris, Nov. 1989

[PATT90] Patterson D.A., Hennessy J.L,
"Computer Architecture: a quantitative approach",
Morgan Kaufmann Publishers, inc. 1990

[POTT83] Potter J.L.,
"Image Processing on the Massively Parallel Processor",
Computer, Jan.1983, pp.62-67

[PRC89] Compte rendu des Premières Journées du PRC Architecture de Machines Nouvelles,
Grenoble, les 8 & 9 Novembre 1989

[PRES83] Preston KJ,
"Cellular Logic Computers for Pattern Recognition",
Computer, pp.36-47, Janvier 1983

[QUIN89] Quinton P, Robert Y,
Algorithmes et Architectures Systoliques,
Masson, ERI, 1989

[ROBE65] Roberts LG,
"Machine perception of three-dimensional solids",
Optical and Electro-optical Information Processing, Tippett et al. Eds, MIT Press, 1965

[ROUX90] Le Roux J,
"Réseaux d'interconnexion multiétages: simulations et essais de modélisation",
TSI, vol. 9, No 2, 1990

[SCHW87] Schwederski T, et al,
"Design and implementation of the PASM prototype control hierarchy",
2nd Intern. Supercomputing conference, pp.418-427, May 1987

[SIEG90-1] Siegel HJ, Nation WG, Allemang MD,
"The organization of the PASM reconfigurable parallel processing system",
To appear in the Proc. of the 1990 Parallel Computing Workshop, OHIO State University, 1990

[STAM75] Stamopoulos CD,
"Parallel Image Processing",
IEEE Trans. on Comp., vol C-24, No 4, pp.424-433, April 1975

[STER81] Sternberg SR,
"Parallel Architectures for image processing",
Real-time/Parallel Computers: Image Processings, Onoe et al, eds, Plenum Press, NewYork, 1981

[TUCK88] Tucker LW, Robertson GG,
"Architecture and applications of the Connection Machine",
Computer, pp.26-38, August 1988

[ZAVI86] Zavidovique B,
Du traitement des images en temps réel à la vision par ordinateur,
Semaine intern. de l'image électronique, CESTA, Nice, Avril 1986

"Et cette absence, dans ma vision, des démarcations que j'établirais bientôt entre elles, propageait à travers leur groupe un flottement harmonieux, la translation continue d'une beauté fluide, collective et mobile."

M. Proust

chapitre 2

L'extraction de segments

Ce chapitre concerne l'extraction de primitives pour la machine Pastis. Dans une première partie, nous rappelons l'intérêt de l'extraction de segments dans des dessins au trait, et plus particulièrement de segments courts. Le but de l'exposé est clairement précisé: trouver une méthode d'extraction de segments qui soit adaptée à nos contraintes de temps et de complexité architecturale (pour une intégration silicium plus facile).

Dans une seconde partie, nous analysons de manière critique les différentes méthodes existantes, qu'elles soient analytiques, morphologiques ou connexionnistes. Bien que la méthode analytique apporte le plus d'information, son utilisation au niveau le plus bas de l'image ne peut s'envisager si l'on a fixé des contraintes de temps strictes. Les méthodes morphologiques semblent coûteuses à mettre en oeuvre complètement. Quant aux méthodes connexionnistes, les tentatives effectuées se sont révélées désastreuses.

Dans une troisième partie, nous proposons notre méthode. Celle-ci est comparée à ce que propose D.Walter; nous expliquons pourquoi les propositions de mise en oeuvre, faites par celle-ci, ne répondent pas à nos choix d'évaluation de méthodes et d'évolution de la machine. Les simulations effectuées sont présentées; elles dénotent l'intérêt de la méthode d'extraction approchée de segments.

chapitre 2

L'extraction de segments

1 Le projet PASTIS

1.1 Présentation d'une certaine démarche

Le projet PASTIS a débuté en Octobre 1988 avec un double objectif. Il s'agissait d'étudier le comportement d'algorithmes réputés difficiles, lorsqu'on leur trouvait de 'bonnes' approximations dans un domaine restreint: la reconnaissance (extraction de primitives, classement,...) et l'apprentissage (nommage, création de nouvelles classes,...) de dessins au trait. Le deuxième objectif était de lier cette réflexion algorithmique à la conception d'une architecture adaptée permettant un fonctionnement rapide de cette machine de Vision. Très pragmatique dans son essence, et sans avoir jamais cherché à l'imposer *ex cathedra*, notre projet tendait à unifier deux forces actuelles de l'informatique: la recherche d'algorithmes plus efficaces par approximation d'algorithmes coûteux, l'utilisation des nouvelles techniques (VLSI, ASIC) pour la conception d'architectures dédiées. Conscient des liens que tissent les algorithmes et l'architecture, le projet PASTIS nous a servi de plate-forme expérimentale, révélatrice de leur importance.

En pleine ébullition connexionniste, nos efforts d'alors portèrent sur l'examen des réalités de telles méthodes appliquées à la Vision. Il nous apparut rapidement que le domaine de la Vision ne pouvait que souffrir d'une inadéquation (pour l'instant) de ces méthodes, essentiellement due à leur apprentissage de type pixel. Beaucoup de démonstrations ont prouvé que si les "réseaux neuronaux" sont capables d'apprendre une image, voire plusieurs, ils n'*apprennent* qu'à classifier l'espace de représentation qu'on leur présente. Cette simple classification sans intervention (*unsupervised learning*) peut sembler suffisante pour des domaines précis où le modèle du monde est clos et connu *a priori*; elle devient insuffisante si l'on étend l'usage de la machine à un monde inconnu, ou en évolution permanente. Nous reviendrons par la suite sur les modèles connexionnistes, et leur comportement.

Néanmoins les idées fortes du connexionnisme ont été conservées: approximation, rapidité, parallélisme du type grain fin, et évaluation globale de traitements locaux. Ces idées sont sous-jacentes à notre réalisation.

1.2 Domaine étudié

Comme nous l'avons indiqué dans l'introduction, la vision embrasse un spectre large d'activités, large par l'éloignement des problèmes et des solutions, mais aussi par le type d'images traitées, l'objectif des différents systèmes de vision et les moyens mis en oeuvre pour l'atteindre. Nous profitons donc de ce paragraphe pour définir clairement ces trois points importants: "l'objet analysé, pourquoi, et comment".

1.2.1 Le dessin au trait

Nous restreignons le domaine d'étude à celui du dessin au trait. Par dessin au trait nous entendons les dessins réalisés au crayon, pour nous avec une souris, comportant des segments, des arcs de cercle,..., c'est à dire toutes les figures que l'on peut retrouver dans les croquis. Nous n'incluons aucunement les textures, les couleurs ou niveaux de gris. L'image est donc composée d'un ensemble de points, appelés *pixels*, ayant deux valeurs: noir ou blanc, 1 ou 0. La grosseur du trait sera prise unitaire; la variation de ce paramètre n'engendre pas une remise en cause des résultats acquis.

Nul doute que le dessin au trait soit un excellent terrain d'évaluation de nouvelles méthodes de vision. Néanmoins, et pour ceux qui en douterait -ce domaine restant ouvert et difficile-, [WALT87, p.262] dévoile les idées majeures qui font de ce domaine restreint un tremplin efficace pour le développement d'études de vision quelque peu générales. Ainsi:

> *"In fact there is evidence that humans can name the objects depicted in a line drawing, as fast as the same objects depicted in natural colors photographs"*,

semble un argument indiscutable et qui confirme le recours aux dessins au trait (*line drawing*) pour l'étude des primitives de base de la vision. L'homme n'a certainement pas développé des systèmes parallèles de vision: l'un pour le dessin au trait, l'autre pour les formes 2D, un troisième pour la vision 3D,... Le fonctionnement du système visuel humain dénote une séparation des traitements dans ce que les neurophysiologistes appellent les aires visuelles. Nous conseillons à ce sujet un article intéressant, [THOR88], dont la lecture ne nécessite aucune connaissance particulière.

Ceci dit, la vision est un problème suffisamment difficile pour qu'il n'ait en plus à souffrir de querelles d'Allemand. Aussi la restriction du domaine d'étude, et encore plus l'ensemble des résultats présentés ici, doivent être reçus et compris comme une simple curiosité scientifique, hors de toute conception manichéenne. Au contraire, à l'instar de [SERR86, *cf. son introduction*], nous pensons que la vision aboutira par une fusion insensible des différentes méthodes existantes et qui, par dessus tout, se ressemblent non seulement dans leur finalité, mais aussi dans leur approche, comme le lecteur s'en apercevra au long de cette lecture[1].

1.2.2 Fonctionnement de Pastis

L'étude de l'organisation logicielle de la machine PASTIS, et des simulations intégrant les MAD sont en cours: thèse de Michel Binse. L'organisation de la base de connaissances symboliques, l'utilisation des MAD dans le processus ouvert (protocole de questions-réponses) d'apprentissage ne sont pas encore figées. En effet, l'idée même de notre démarche veut que la définition logicielle de notre machine évolue selon les résultats des simulations successives.

1. D'ailleurs "choisir la meilleure solution" reste un problème NP-dur quand le domaine n'est pas de ceux convexes...

Toutefois il importe de savoir que PASTIS est destiné à l'apprentissage. Cet apprentissage implique selon nous l'usage de processeurs d'extraction d'information et d'analyse, extrêmement puissants. La puissance ici ne se mesure pas en "Flops", mais en adéquation entre le processus mis en oeuvre et l'architecture cible. La rapidité d'exécution des tâches est la condition *sina qua non* d'un apprentissage que nous qualifierions "par essais-erreurs". Cette nécessité relative aux performances du matériel se retrouve en filigrane dans de nombreux projets de vision; nous pensons par exemple à la machine PVV qui fonctionne par "prédiction-vérification" [LUX84]. Quant aux aspects propres de l'apprentissage, nous n'y reviendrons pas dans cette thèse: ils n'en forment pas l'objectif essentiel.

Par contre, et pour situer rapidement notre démarche face aux techniques neuronales, nous avons choisi une approche symbolique, structurelle. L'image contient certains objets, que nous essayerons de caractériser par des primitives fixées à l'avance. La machine ne génèrera pas ses propres primitives de classification, à la manière des réseaux connexionnistes. Elle est conçue pour analyser certaines formes d'objets à l'aide de ces primitives, et donc ne pas "voir" tout ou partie des objets formés à base de primitives complémentaires -si le monde est clos-.

Enfin, la contrainte essentielle à toute réalisation reste l'objectif temps réel. [THOR88] indique que le cerveau humain analyse et reconnaît un objet dans un laps de temps d'environ 100 à 120 milli-secondes, plus proche selon lui de 60 milli-secondes pour la phase concernant le cortex visuel: véritable centre d'analyse et de reconnaissance. Ce temps permet de déduire le nombre d'étapes de traitements effectués; les informaticiens parleraient d'étages de *pipeline*. Ce nombre serait d'une dizaine seulement!

Toute proportion gardée, si les traitements envisagés étaient les bons, en ce sens qu'ils mèneraient à une vision *quasi*-humaine, chacun de ceux-ci devraient s'exécuter en 10 milli-secondes environ... sur plus de cent millions de photorécepteurs par oeil!... L'élément que nous retiendrons est cette valeur: 10 milli-secondes. Elle fixe relativement bien les contraintes de temps auxquelles nous aurons à faire face.

1.2.3 Les primitives retenues

Par *primitives*, nous entendons les traits caractéristiques d'une forme quelconque, et qui permettent de décrire le contenu, ou tout au moins une partie, d'une image. La primitive est une notion essentielle à la vision. En morphologie mathématique, on parle d'élément structurant; en convolution, de noyaux; en filtrage, de fonctions génératrices; en analyse structurelle, d'alphabet ou de code. Si l'on s'intéresse aux paramètres mesurables au sens de la métrique, les primitives utiles au classement des images sont: l'angle, la longueur des cotés, l'aire, le périmètre, la longueur d'arc, et les frontières de l'objet [LEVI85].

Bien entendu, il existe une multitude de primitives possibles, et leur choix dépendra de l'application visée. La texture est une bonne primitive pour détecter des régions. Les segments et arcs de cercle sont deux bonnes primitives pour caractériser les contours. Pour le projet PASTIS, nous ne nous sommes intéressés qu'aux primitives déterminant les contours, puisque le dessin au trait n'est fait que de contours. Bien que l'arc de cercle soit une primitive importante, l'étude présentée ici n'abordera que la caractérisation d'une forme à l'aide d'une primitive de base: le **segment**.

1.3 Les segments orientés de petite taille

Face à la variété extraordinaire des primitives existant en vision, le choix du segment peut paraître suicidaire. Toutefois, dans le domaine du dessin au trait, l'existence de l'approximation polygonale démontre que toute courbe peut se développer comme une suite de segments. De plus, comme le remarque [PAVL77], tout image provenant d'un ordinateur est en réalité un ensemble de segments -nous n'aborderons pas ici l'existence des facettes triangulaires, chères aux techniques de synthèse d'image-. Aussi ceci conforte quelque peu l'idée d'étudier ces segments.

Cependant il faut être particulièrement attentif à la situation même du projet, dans ce microcosme important des extracteurs de segments. En effet, maintes applications existent, surtout en robotique, qui utilisent les segments efficacement que ce soit pour reconnaître des objets, les positionner, ou encore faire de la vision binoculaire, du repérage de cibles -eh oui!-, etc..., sans nullement prétendre à l'exhaustivité. Le segment est à lui seul une sorte de Graal moderne qui confine la recherche de bon nombre de scientifiques: automaticiens, neurophysiologistes, informaticiens. Nous verrons plus loin combien les découvertes sur le cortex visuel du singe corroborent cet intérêt manifeste.

L'essentiel provient de ce que cette description en segments, et surtout la phase d'approximation polygonale qui suit nécessairement dès lors que l'oeil voit des droites et non des suites de segments, sont des problèmes difficiles au sens du calcul tel que nous le connaissons; pourquoi pas NP-complet? Le but de cette étude est de proposer une démarche originale d'extraction de ces segments, qui offrira une description satisfaisante, avec le flou inhérent à ce terme, par des algorithmes *quasi*-linéaires.

Si la recherche d'algorithmes plus performants n'a rien d'original, par contre l'utilisation pêle-mêle de méthodes d'approximation et d'architectures dédiées, présentées au chapitre 3, donnera matière à penser au lecteur, pour relever l'un des grands défis à venir: créer une machine autonome capable de voir un certain côté de notre monde.

2 Les méthodes d'extraction de segments

Nous présentons ici un survol des méthodes existantes et utilisées pour l'extraction de segments. Nous parlons d'abord de l'approximation polygonale, qui reste la méthode la plus employée actuellement dans les machines robotiques de vision. Pour pallier sa complexité d'exécution, la transformée de Hough apparaît comme un pré-traitement possible, de même que la convolution ou le filtrage morphologique. Enfin nous présentons brièvement les propositions des modèles connexionnistes en la matière.

2.1 Approximation analytique

La géométrie analytique permet d'étudier les figures de l'espace par l'algèbre à l'aide des équations des figures ou des coordonnées de leurs points. Le domaine de l'imagerie informatique est gourmand en techniques analytiques:

que ce soit en **synthèse d'images** pour "approcher" des courbes à l'aide de polygones (la plupart des systèmes de synthèse travaillent soit par segments soit par facettes),

ou en **analyse d'images**, où la description polygonale d'une figure autorise sa recherche et sa comparaison dans une base de données, sachant qu'on ne travaille plus que sur quelques équations en remplacement d'une séquence importante de pixels.

Notre propos ne concerne ici que l'analyse d'images, mais le lecteur retrouvera ces techniques dans le domaine de la synthèse. Concernant la synthèse, ces techniques sont utilisées pour dessiner une courbe continue en partant de plusieurs points définis comme des points de cassure, i.e. des points par lesquels la courbe passe ou sur lesquels elle "s'appuie". A l'opposé, dans le domaine de l'analyse d'images, on part d'une courbe définie par une suite de pixels dont on n'a que faire. L'objectif de l'approximation polygonale est d'extraire de cette suite de points une description en plusieurs segments de taille variable, qui pourront servir d'identifiants de la courbe. C'est la phase de vectorisation.

2.1.1 Approximation et Interpolation de courbes

[PAVL82] consacre trois chapitres aux problèmes d'interpolation ou d'approximation dont il précise clairement dès le début la différence:

> *"Finding a curve that passes through a set of given points is the problem of interpolation, while finding a curve that passes near a set of given points is the problem of approximation".*

Nous consacrons les deux paragraphes suivants à ces méthodes: d'abord l'approximation polygonale, puis l'interpolation par les B-splines. Nous montrerons d'autre part comment la transformée de Hough permet de retrouver les équations d'une courbe, en transposant le problème difficile de la recherche du meilleur segment en un problème plus facile de détection de pics dans un tableau d'accumulation.

Pour chacun des sujets abordés, et c'est là notre souci, nous essayerons de préciser quelle est la quantité de calculs requis, quelle peut être la qualité des résultats acquis, et surtout si la recherche de la meilleure approximation ou interpolation converge rapidement ou non.

2.1.2 *L'approximation polygonale*

La donnée du problème est la suivante: soit $E=\{(x_i,y_i)\}$ un ensemble de n points décrivant une figure F, trouver un polygone ou une ligne polygonale[1] P qui approche *au mieux* F tout en ayant un nombre minimum de sommets.

Le terme 'au mieux' doit bien évidemment faire référence à une fonction mathématique que l'on puisse évaluer. L'évaluation doit rendre compte d'une notion de distance entre deux courbes: la figure F et son approximation P. La distance entre un point de F et l'approximation P se calcule par la normale en ce point; une fonction de coût doit regrouper toutes les valeurs de distance pour l'ensemble des points de F. Les deux fonctions **erreur** les plus classiques sont l'erreur maximale ou l'erreur au sens des moindres carrés.

Si err_i correspond à l'erreur calculée au $i^{ème}$ point de E, alors on obtient:

$$\textit{Erreur maximale} \qquad E_{max} = max_i|err_i|$$

$$\textit{Erreur aux moindres carrés} \qquad E_{mc} = \sum_i err_i^2$$

Approximation au sens des moindres carrés

Soit $E=\{(x_i,y_i),\ i=0,1,...,n\}$ l'ensemble des points image, et soit une courbe approximative $g(x)$ définie comme suit:

$$g(x) = \sum_{j=0}^{m} a_j \cdot b_j(x)$$

où les $b_j(x)$, $j=0,...m$, forment une famille de courbes libre et génératrice, i.e. une base pour l'ensemble des courbes qui nous intéressent.

L'erreur err_i calculée pour le $i^{ème}$ point de notre courbe E à approximer est égale à:

$$err_i = y_i - \sum_{j=0}^{m} a_j \cdot b_j(x_i)$$

et

$$E_{mc} = \sum_{i=0}^{n} \left[y_i - \sum_{j=0}^{m} a_j \cdot b_j(x_i) \right]^2$$

L'indépendance linéaire des fonctions $b_j(x)$ est primordiale dans la recherche d'une solution au sens des moindres carrés. Le théorème de projection intervient à cet effet:

- *Théorème de projection:* Soit F un sous-ensemble non vide, convexe, fermé, d'un espace de Hilbert E, alors pour tout élément x de E, il existe un et un seul élément y de F tel que: $\|x - y\| = d(x, F) = p_F(x)$.

1. un polygone est par définition fermé, ce qui n'est pas forcément le cas de notre figure.

La famille libre des $b_j(x)$ forme un sous-espace vectoriel de E, c'est à dire un sous ensemble non vide convexe. On rappelle à ce propos qu'une partie U d'un espace vectoriel E est dite *convexe* si, toutes les fois qu'elle contient deux points u et v, elle contient le segment fermé $[u,v]$ qui les joint; ce qui est le cas pour un sous espace vectoriel. Ainsi F convexe fermé peut se transposer en F sous-espace vectoriel complet.

Il est possible de modifier les équations précédentes pour montrer comment s'applique ce théorème.

et soit $B_X \in M_{\Re}(n,m)$

la matrice définie de la façon suivante: $B_X = \begin{bmatrix} b_0(x_0) & b_1(x_0) & \dots & b_m(x_0) \\ b_0(x_1) & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ b_0(x_n) & \dots & \dots & b_m(x_n) \end{bmatrix}$

Soit $E=\{(x_i,y_i),\ i=0,1,\dots,n\}$ l'ensemble des points image,

on cherche le vecteur A, qui forme la combinaison linéaire $\Sigma a_j b_j(x_k)$ la plus proche du point y_k, pour chacun des k de l'intervalle [0,n]. On cherche A tel que:

$$inf_{A' \in \Re^m} \| Y - B_X \cdot A' \| = \| Y - B_X \cdot A \| = E_{mc}$$

$Im(B_X) = \{U \in \Re^n / U=B_X.A$, pour tout $A \in \Re^m\}$ est un sous-espace vectoriel de $\Re^n$, qui plus est complet puisque $\Re^n$ est complet. Donc quelque soit le vecteur Y de $\Re^n$, il existe un unique vecteur $U=B_X.A$ de $Im(B_X)$ tel que ce vecteur U soit la projection de Y sur $Im(B_X)$. L'unicité de la solution provient de ce que les $b_j(x)$ forment une famille libre. CQFD.

Méthode approchée

La recherche d'une meilleure valeur au sens des moindres carrés est généralement un problème difficile qui nécessite un nombre important de calculs: résolution d'un problème de programmation linéaire ou utilisation d'un algorithme itératif [PAVL82]. On est donc tenté de découper le problème en plusieurs parties, avec comme objectif la recherche d'une droite de régression linéaire qui approche un ensemble de pixels. Hélas cette méthode est sensible au choix des différentes suites de pixels considérées, notamment quant à la position des points de cassure de notre courbe.

Pour faciliter cette recherche, on recourt généralement à des heuristiques approchées qui sont souvent beaucoup moins longues et coûteuses que les algorithmes qui donnent *la* meilleure solution. Les applications robotiques se satisfont plus des algorithmes linéaires ou *quasi*-linéaires. Dans le cas particulier de l'approximation polygonale, l'heuristique la plus employée est appelée: *split-and-merge*, c'est à dire découper-et-fusionner.

Cette heuristique se donne pour objectif de vérifier parmi une suite de points leur colinéarité de manière approchée. S'ils ne sont pas colinéaires, alors la suite est séparée, découpée, jusqu'à l'obtention de la condition de colinéarité. Dans le cas contraire, certaines suites sont fusionnées si leur suite résultante vérifie le critère de colinéarité approximative.

Le critère de colinéarité approximative est simple: il correspond au tracé d'une corde entre deux points, et la vérification que tous les points situés entre ces deux extrêmes ne sont distants de cette corde que d'au plus une distance D. [PAVL82] rappelle l'intérêt de cette approximation ainsi que le sens donné

à cette distance d'un point donné par rapport aux équations des courbes approximantes.

Bien qu'approché, le résultat de cet algorithme est une suite de segments qui possède la propriété suivante [PAVL82]:

❑ *Proposition:* Le nombre de segments obtenus par un algorithme du type découper-et-fusionner est toujours inférieur à deux fois le nombre minimum de segments.

▲ *Preuve:* Soient I_1, I_2,...I_n les intervalles de la partition minimale et J_1, J_2,...J_p ceux trouvés par l'algorithme découper-et-fusionner. Aucun intervalle I_i ne peut contenir plus d'un intervalle J_k; dans le cas contraire, J_k et J_{k+1} auraient forcément fusionné. Par contre, chaque I_i peut contenir à l'intérieur de ces bornes l'intervalle J_k et une partie des intervalles J_{k-1} et J_{k+1}, recouvrant aussi I_{i-1} et I_{i+1} respectivement.
On a donc au plus n intervalles J_k à l'intérieur de chacun des intervalles I_i de la partition minimale, ainsi qu'autant d'intervalles J_k qu'il n'y a de points de séparation dans cette partition minimale,
c'est à dire: n+n-1. D'où $p \leq 2n-1$.

Algorithmes

Les algorithmes d'approximation polygonale possèdent plusieurs critères d'optimalisation difficiles à satisfaire complètement. Ils concernent:

(i) le maximum d'erreur obtenu sur chacun des segments,
(ii) le nombre minimal de segments que possède la description,
(iii) le périmètre du polygone minimal, et
(iv) un temps d'exécution raisonnable de l'algorithme.

[RAME72] présente un algorithme de *découpe*, que [BANG86] considère comme l'un des plus utilisé en robotique. Nous le donnons ci-dessous:

---

1. Choisir *n* points, $n \geq 2$, qui forment un polygone de départ
2. Pour chaque segment du polygone courant **faire**
    Chercher le point image *P* de distance maximale, d_{max}
    Si $d_{max} >$ *seuil* **alors**
        P est un nouveau sommet du polygone
        le coté considéré est divisé en 2 cotés en *P*
    finsi
finpour

Cet algorithme se retrouve sous diverses formes dans les applications robotiques [DELO88]. Les modifications apportées concernent surtout la poursuite de l'approximation par une corde tant que les points restent alignés sur cette corde. De fait les points choisis au départ ne sont pas forcément les points de cassure obtenus à la fin de l'algorithme. [LUX84] développe un tel algorithme qui lui permet

d'obtenir une liste finale simple de directions de droites pour le système de vision PVV:

---

1. Recherche de p points alignés à un seuil d'erreur δ_1 près:

- prendre p points image successifs de la liste ordonnée des éléments de contour: $p_i...p_{i+p}$
- tracer la corde $[p_i, p_{i+p}]$
- chercher p_{max} tel que $d_{max} = d(p_{max}, corde) = sup(p_k, corde)$, $k=i...i+p$
- **si** $d_{max} > \delta_1$ **alors** $p_i = p_{max}$

  aller en 1
- **finsi**

2. Recherche des points suivants alignés à un seuil δ_2 près:

- construire une bande d'épaisseur δ_2 parallèle au segment issu des p points précédents
- les points suivants $p_{i+p+1}...p_f$ appartenant à l'intérieur de cette bande appartiennent au même segment de droite
- $p_i = p_f$
- **aller en 1.**

[BANG86] présente une application robotique de l'approximation polygonale. L'algorithme d'approximation crée une description de l'image sous forme d'un codage de Freeman, que nous présentons de suite. L'application robotique nécessite un temps d'exécution faible, et une bonne stabilité de la représentation obtenue. Aussi les différents exemples d'application de cette méthode dérivent le plus souvent un algorithme itératif, comme celui présenté ci-dessus, en fonction des contraintes propres de l'environnement [AYAC83].

2.1.3 Le code de Freeman

A la fin des années soixante, Freeman introduisit le codage de chaînes, qui consiste en une suite de segments définis sur une grille avec un ensemble fini d'orientations possibles. Depuis référencée sous le nom de code de Freeman, cette structure permet de représenter efficacement les dessins au trait, à cause des contraintes strictes supportées par la construction [BALL82]. Un seul point est défini par sa position, le point de départ; la suite de la courbe est représentée par déplacements successifs sur la grille. La distance entre points importe peu, puisque la grille est uniforme et connecte les points entre eux aux quatre ou huit premiers voisins; seule l'orientation est une information à retenir, codée sur 2 ou 3 bits.

Comme le remarque [BALL82] et l'atteste [BANG86], le codage de Freeman est particulièrement apprécié pour les phases de fusion des algorithmes d'approximation polygonale. Ainsi le code dérivé, qui code les variations d'orientation, permet de repérer rapidement les segments à fusionner; ceci se comprend naturellement, nous ne l'approfondissons pas. Appliqué à l'analyse structurelle, [MICL84] donne un exemple d'utilisation du code de Freeman pour la reconnaissance des caractères Coréens. Dans un arbre, les feuilles représentent des formes primitives notées avec ce code.

Voici à titre indicatif, le code de Freeman d'une courbe quelconque:

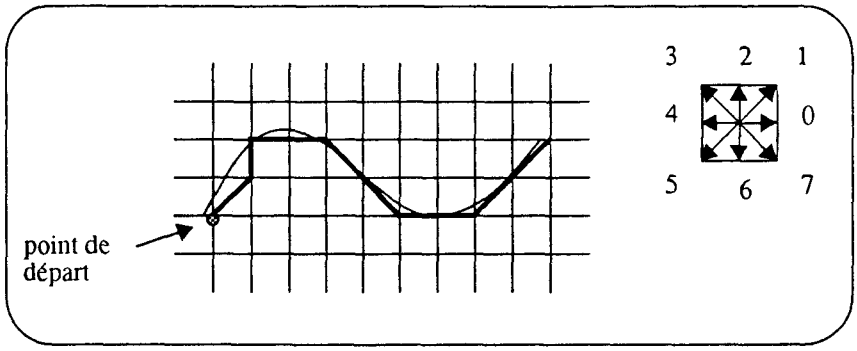


figure 1 codage de Freeman, la courbe possède pour code le mot a=1200770011

Etendu au continu, le code de Freeman se dérive pour donner naissance aux ψ-*s* courbes [BALL82]. Celles-ci donnent directement la segmentation d'une courbe, si les points de changements d'orientation ne sont pas trop nombreux. Le lecteur intéressé trouvera dans [JUVI84] une description détaillée d'une application robotique de vision basée sur le codage de Freeman, associée aux problèmes d'approximation et de reconnaissance.

2.1.4 Les B-splines

Le terme *spline* nous vient d'une époque où n'existaient pas encore les machines informatiques. Afin de tracer une courbe lisse, les dessinateurs utilisaient une règle en bois flexible, appelée une *spline*, et des poids. Ils posaient un poids sur chacun des points de leur courbe, et venaient caler la règle contre ces poids à l'aide des protubérances que comportaient ceux-ci. Ainsi obtenait-on une courbe aux dérivées première et seconde continues. Bien qu'aujourd'hui, les B-splines soient *n* fois dérivables de dérivée continue, ce nom est resté.

Ce petit rappel nous permet de comprendre exactement ce qu'est une fonction polynomiale par morceaux. On peut très bien imaginer cette règle fixée aux points sur la planche à dessin: l'un des avantages liés à cette méthode consiste en la relative localité des modifications apportées à la courbe lors du déplacement d'un des points (d'un des poids). Bien entendu les B-splines présentent d'autres avantages, mais avant d'y revenir nous allons décrire ce qu'est une fonction polynomiale par morceaux.

Fonction polynomiale par morceaux

Soient $x_1, x_2,..., x_{k-1}$ une suite de points qui séparent l'intervalle $[a,b]$ en k intervalles, et qu'on a coutume d'appeler des *breakpoints* (points de cassure). On suppose que $x_0=a$ et $x_k=b$. La donnée d'une fonction polynomiale par morceaux est alors la suivante:

$$p(x) = p_i(x) \qquad \text{où } x_i \le x \le x_{i+1} \qquad \text{pour } i=0,1,...,k-1 \qquad (3.1)$$

$$p_i^{(j)}(x_i) = P_{i+1}^{(j)}(x_i) \qquad \text{pour } j=0,1,...,r-1 \quad \text{et } i=1,2,...,k-1 \qquad (3.2)$$

Les fonctions $p_i(x)$ sont des polynômes de degré m ou moins. Leur continuité aux points de cassure est exprimée par l'équation (3.2), où la valeur de r peut être nulle, induisant de fait aucune contrainte quant à la continuité des dérivées du polynôme (pour $r=0$, on joint les breakpoints par des segments sans lissage local). Le cas où $r=m=3$ fait directement référence à la règle dont nous avons déjà parlée.

Définition 2.1: Une fonction *spline simple* est une fonction polynomiale par morceaux décrite par les équations (3.1) et (3.2) avec $r=m$. Pour r=1, 2 ou 3, on parle de spline linéaire, quadratique ou cubique.

Définition 2.2: Une spline est une fonction polynomiale par morceaux décrite par les équations (3.1) et (3.2) avec $r < m$.

Les fonctions splines et B-splines sont fréquemment utilisées en synthèse d'images. La référence en la matière est le chapitre 11 de [PAVL82], qui donne les équations de ces B-splines. L'avantage principal des fonctions splines concerne la localité des modifications apportées à la courbe lors du déplacement d'un point de cassure. Appliquées à la description de formes, les fonctions splines de degré n (n=3 généralement) encadrent la courbe, celle-ci devant tenir dans les polygones convexes formés par les suites de n+1 points consécutifs comme le montre la figure 2:

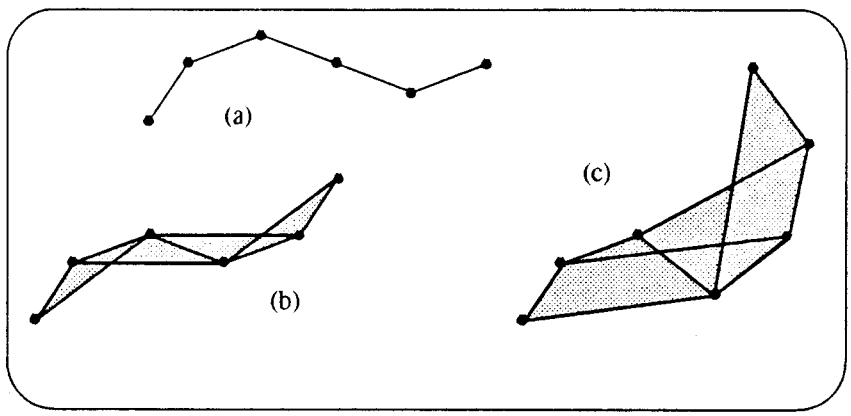


figure 2 la fonction de degré n doit tenir dans les polygones convexes formés par les suites de n+1 points consécutifs. (a) n=1, linéaire, (b) n=2, quadratique, (c) n=3, cubique.

La recherche s'effectue sur la position de ces points de cassure; le caractère local de l'influence de ces points facilite l'obtention d'une description. Pour plus de détails, [BALL82, pp.239-243] présente l'utilisation des B-splines comme méthode de description de figures.

Nous abandonnons ici les fonctions splines, en remarquant que ces fonctions sont complexes à calculer, car généralement d'un degré important, et que la description obtenue est certainement trop précise: la recherche de la bonne position des points de cassure, sans parler de leur nombre, fait de cette méthode une sorte d'approximation polygonale raffinée. Au contraire, maintenant nous allons nous intéresser aux méthodes qui permettent d'obtenir des informations quant à la position des points de cassure ou la présence de droites, sans garantir ces informations comme une approximation polygonale. Il s'agit donc d'étudier les méthodes qui permettent d'accélérer cette approximation en obtenant des renseignements essentiels à la maîtrise de sa complexité.

2.2 La transformée de Hough

2.2.1 Introduction

La transformée de Hough fut introduite pour la première fois par [HOUG62] pour détecter des formes complexes de points dans une image binaire. L'originalité de cette méthode est de transposer la description de ces formes complexes dans un espace de paramètres (quantifiés) qui les caractérisent. Ainsi, si l'on veut extraire les paramètres d'une ligne passant par un ensemble de points donnés de l'image à analyser, on produit avec chacun de ces points des vecteurs caractéristiques de toutes les droites qui passent par ce point.

S'il existe effectivement une corrélation linéaire entre les points analysés, la disposition des vecteurs obtenus donne un couple de paramètres (pente, ordonnée à l'origine) de la droite. La transformée de Hough transpose donc la donnée d'un problème difficile[1] en celle d'un problème plus facilement soluble: la recherche de pics locaux dans l'espace des paramètres [ILLI88].

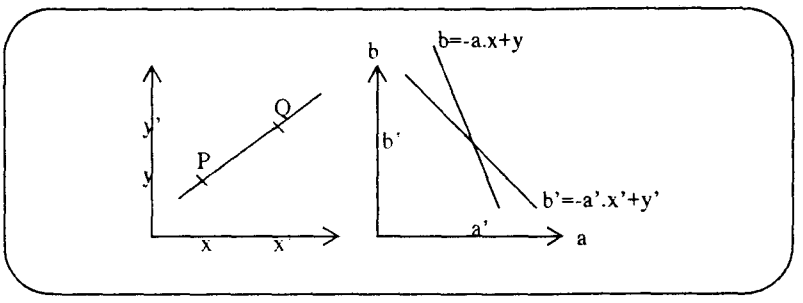


figure 3 La ligne dans l'espace image, puis dans l'espace des paramètres

Soit le point P(x,y) de la figure 3, et l'équation d'une droite: y=a.x+b.

Si l'on quantifie l'espace de représentation des paramètres *a* et *b*, et si l'on cherche toutes les droites passant par P, alors

la réponse correspond à l'ensemble des points *(a,b)* qui satisfont: $b=-a.x+y$.

On est donc passé d'une ligne dans l'espace image à une ligne dans l'espace des paramètres. En réitérant ce processus, on peut évaluer pour tous les points de l'ensemble de départ les valeurs des paramètres de leurs droites possibles. Pour deux points P(x,y) et Q(x',y') il peut exister une intersection entre les droites de paramètres (a,b) et (a',b').

Ainsi tous les points *(x,y)* qui sont colinéaires dans l'espace image vont générer des droites dans l'espace de paramètres qui se couperont en un point commun *(a,b)*. Ce point caractérise la ligne de l'espace image qui porte les points *(x,y)*. La transformée de Hough identifie ces points d'intersection dans l'espace des paramètres. Cette détermination est une opération locale, bien plus simple à mettre en oeuvre que les heuristiques d'approximation polygonale traitées.

Pour détecter les points d'intersection, on utilise une *matrice d'accumulation* de dimension celle de l'espace de paramètres (espace quantifié au départ). Chaque élément *(a,b)* de cette matrice contient le nombre (accumulation) de points *(x,y)* de l'image à analyser ayant satisfait à l'équation: $b=-a.x+y$.

1. nous avons vu que l'approximation polygonale est difficile au sens où l'obtention d'une meilleure solution nécessite maints calculs.

On peut alors dériver l'algorithme suivant [BALL82]:

(i) Quantifier l'espace des paramètres entre les valeurs minimales et maximales de *a* et *b*.

(ii) Créer une matrice *M(a,b)* dont les éléments sont à zéro.

(iii) Pour tout point *(x,y)*, incrémenter tous les points de *M*,

$$M(a,b) = M(a,b) + 1$$

pour tous les *couples (a,b)* satisfaisant: $b=-a.x+y$.

(iv) Les maxima locaux de M correspondent à des points colinéaires de l'image. Leur valeur fournit une information sur le nombre de points de la ligne

---

Cette technique s'étend facilement à la détection d'autres types de courbes paramétriques. Mais, comme le rappelle [BALL82], le calcul et la taille de la matrice d'accumulation croît de façon exponentielle avec le nombre de paramètres. Ainsi pour la détermination de n paramètres échantillonnés chacun en p intervalles, la matrice d'accumulation possède p^n éléments. La transformée de Hough est donc naturellement limitée à des courbes simples: c'est là son gros inconvénient.

2.2.2 *Généralisation de la méthode*

Un ensemble de points image *(x,y)* qui sont portés par une droite peut être défini par la relation *f*, telle que:

$$f((\hat{a},\hat{b}),(x,y)) = y - \hat{a} \times x - \hat{b} = 0$$

où a est la pente et b l'ordonnée à l'origine de cette droite. La transformée de Hough évalue les paramètres *(a,b)* de toutes les droites qui passent par un point *(x,y)* de l'image. Il s'agit donc d'une application de l'espace image vers l'espace de paramètres, définie par la relation duale (*g(.)* est l'application duale de *f(.)*):

$$g((\hat{x},\hat{y}),(a,b)) = \hat{y} - a \times \hat{x} - b = 0$$

L'extension de la TH[1] aux courbes autres que les droites, est définie par l'équation:

$$g((\hat{x},\hat{y}),(a_1, a_2, \ldots, a_n)) = 0$$

où la courbe de référence est caractérisée par n paramètres a_1, a_2,... a_n. Cette relation duale définit une hypersurface dans l'espace des paramètres de dimension n. La courbe qui porte les points de l'image est obtenue par l'intersection des hypersurfaces dans l'espace des paramètres.

---

1. TH = Transformée de Hough

Extension à tout type de formes

La TH s'étend aux formes quelconques, non analytiques. Dans la TH généralisée, une forme recherchée est représentée par un échantillon de base défini par la liste des points frontière de la forme: $\{(u_i,v_i)\}$ pour $i=1,...N$. Cette liste est une instance particulière d'une classe de points frontière d'une forme donnée. D'autres instances d'une classe, c'est à dire d'autres échantillons, peuvent être obtenus par similitudes, en effectuant translation, rotation ou homothétie sur la forme de base de la classe (cf. figure 4).

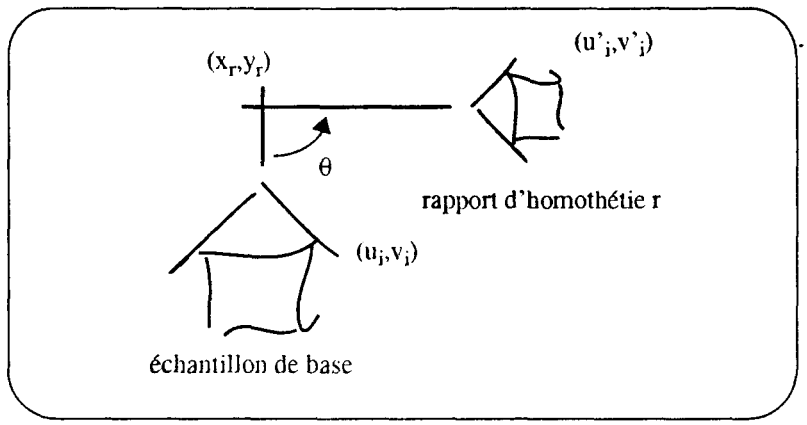


figure 4 homothétie et rotation de l'échantillon de base de la classe.

Les nouveaux points frontière obtenus par homothétie et rotation de notre forme de base sont définis par:

$$\begin{cases} u'_i = x_r + ((y_r - v_i) \cdot r \cdot \sin\theta) - ((x_r - u_i) \cdot r \cdot \cos\theta) \\ v'_i = y_r - ((x_r - u_i) \cdot r \cdot \sin\theta) - ((y_r - v_i) \cdot r \cdot \cos\theta) \end{cases}$$

Les deux paramètres r et θ décrivent la transformation entre les deux échantillons. Une translation relative à (x_r,y_r) peut s'exprimer en ajoutant deux autres paramètres (x_{tr},y_{tr}), ce qui donne:

$$\begin{cases} u''_i = u'_i - x_{tr} = x_r + ((y_r - v_i) \cdot r \cdot \sin\theta) - ((x_r - u_i) \cdot r \cdot \cos\theta) - x_{tr} \\ v''_i = v'_i - y_{tr} = y_r - ((x_r - u_i) \cdot r \cdot \sin\theta) - ((y_r - v_i) \cdot r \cdot \cos\theta) - y_{tr} \end{cases}$$

Si un point image (x_j,y_j) appartient à cet échantillon, alors on a $x_j - u''_i = 0$ et $y_j - v''_i = 0$, pour l'un des points i de l'échantillon. La TH permet d'évaluer toutes les combinaisons d'un point image et d'un point échantillon; et en utilisant les équations précédentes de déterminer les valeurs r, θ et (x_{tr},y_{tr}), qui transforment l'échantillon de base en un échantillon candidat. Bien évidemment ce calcul devient trop important, et on essayera de fixer les dimensions de l'échantillon considéré, par des valeurs connues de r et θ. Dans ce cas, il faut déterminer le couple (x_{tr},y_{tr}), tel que:

$$\begin{cases} x_{tr} = x_j + x_r - u'_i = x_j - ((y_r - v_i) \cdot r \cdot \sin\theta) + ((x_r - u_i) \cdot r \cdot \cos\theta) \\ y_{tr} = y_j + y_r - v'_i = y + ((x_r - u_i) \cdot r \cdot \sin\theta) + ((y_r - v_i) \cdot r \cdot \cos\theta) \end{cases}$$

Définition formelle de la transformée de Hough

Pour une forme analytique ou une forme échantillonnée $\{(u_i,v_i)\}$, la TH d'un ensemble de points image (x_j,y_j), $j=1...L$ se définit de façon formelle comme suit:

Soit $g(\hat{x}_j, \hat{y}_j, a_1, \ldots, a_n)$ l'application de l'espace image dans l'espace des paramètres,

i.e.

$$g = \begin{cases} f(\hat{x}_j, \hat{y}_j, a_1, \ldots, a_n) = 0 & \text{forme analytique} \\ \prod_{i=1}^{N} \left\{ [x_{1r} - (\hat{x}_j + x_r - u_i)]^2 + [y_{1r} - (\hat{y}_j + y_r - v_i)]^2 \right\} = 0 & \text{forme échantillonnée} \end{cases}$$

alors la TH, $H(a_1,\ldots,a_n)$ est définie comme

$$H(a_1, \ldots, a_n) = \sum_{j=1}^{L} h(\hat{x}_j, \hat{y}_j, a_1, \ldots, a_n)$$

$$h(\hat{x}_j, \hat{y}_j, a_1, \ldots, a_n) = \begin{cases} 1 & si \quad g(\hat{x}_j, \hat{y}_j, a_1, \ldots, a_n) = 0 \\ 0 & autrement \end{cases}$$

Remarques

D'un point de vue algorithmique, nous avons vu que l'espace des paramètres est quantifié; ceci correspond à une décomposition de cet espace en un nombre fini de régions bornées. Chacune de ces régions est alors associée à une entrée de la matrice d'accumulation. Cette entrée s'apparente à un compteur incrémenté quand une hypersurface (donnée duale d'un point de l'image) traverse la région qui lui est associée [ILLI88]. Ceci explique le nom de matrice d'accumulation.

La recherche des pics dans la matrice d'accumulation est plus facile et moins coûteuse en calcul qu'une recherche de formes spécifiques au niveau pixel; le terme anglais est ici plus parlant: *template matching*. Une telle recherche nécessite l'évaluation de la correspondance entre l'échantillon et une zone de l'image, au niveau pixel, et la répétition de cette évaluation pour chaque déplacement de l'échantillon sur l'image. Le *template matching* extrait donc les zones qui s'apparentent au mieux avec l'échantillon. Cependant, la TH permet de modéliser un échantillon *déformable*; les paramètres ne sont pas figés, mais évoluent dans un intervalle donné par la matrice d'accumulation. Le *template matching* ne se conçoit qu'avec un échantillon précis, si l'on espère obtenir une information fiable.

[ILLI88] rappelle les avantages de la TH. D'abord elle permet la reconnaissance de formes partiellement déformées, ce qui est un avantage surtout dans le cas d'objets en partie cachés. Ensuite, corollaire, elle s'avère *peu sensible au bruit*, propriété importante si l'image subit d'abord une phase de segmentation.

Enfin, et c'est là l'intérêt premier de cette méthode, le calcul pour tous les points de l'image est *parallélisable*, de même que la recherche simultanée de plusieurs courbes dans le cas où celles-ci n'interfèrent pas dans l'espace des paramètres.

Coordonnées polaires

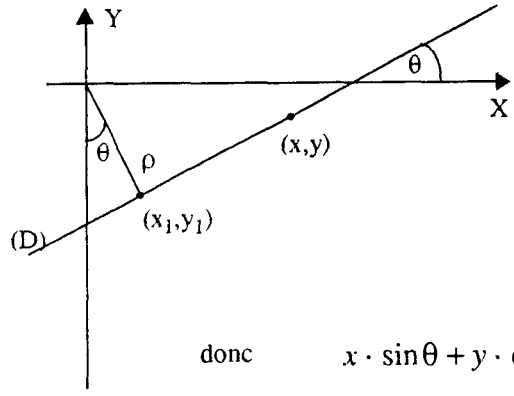
Les équations de la TH font référence aux coordonnées cartésiennes, ce qui empêche la modélisation des lignes de pente très grande. Aussi [DUDA72] introduit l'utilisation des coordonnées polaires pour la TH, où ρ est la longueur et θ l'angle du vecteur normal de cette ligne par rapport au point origine du référentiel de l'image:

Soit (x,y) un point image, il passe par ce point une infinité de droites.

Soit D, l'une de ces droites, alors

de la figure ci-dessous, il vient: $(x - x_1)\sin\theta + (y - y_1)\cos\theta = 0$

$$x \cdot \sin\theta + y \cdot \cos\theta = x_1 \cdot \sin\theta + y_1 \cdot \cos\theta$$



or $\cos\theta = \frac{y_1}{\rho}$ et $\sin\theta = \frac{x_1}{\rho}$

donc $x \cdot \sin\theta + y \cdot \cos\theta = x_1 \cdot \frac{x_1}{\rho} + y_1 \cdot \frac{y_1}{\rho} = \frac{x_1^2 + y_1^2}{\rho} = \frac{\rho^2}{\rho} = \rho$

ce qui donne l'équation: $\boxed{x \cdot \sin\theta + y \cdot \cos\theta - \rho = 0}$

L'utilisation des paramètres (ρ,θ) signifie que les points image sont transposés en courbes sinusoïdales dans l'espace des paramètres. L'avantage de cette représentation vient de ce que les paramètres n'ont plus une excursion infinie comme pour les coordonnées cartésiennes lorsque la pente devient très grande. Dans la TH, ceci réduit l'espace quantifié des paramètres. Dans le cas des lignes, l'espace représente sur un axe la distance perpendiculaire des droites extraites par rapport au centre de l'image, et sur l'autre axe la pente de ces droites par l'angle qu'elles font avec leur normale au centre de l'image.

2.2.3 *Efficacité de la Transformée de Hough*

Les études effectuées sur l'efficacité de la TH concernent essentiellement deux domaines distincts: l'amélioration des techniques d'accumulation pour réduire les coûts de stockage (taille de matrice) et de calcul, et l'impact de la distribution des paramètres sur la forme des pics créés (pics prononcés et isolés). Ce dernier problème, nous le retrouverons dans le fonctionnement de Pastis; il rejoint le choix de paramètres, dont dépend la qualité des informations extraites.

Influence des paramètres

Chronologiquement, [COHE77] fut le premier à évaluer l'efficacité de la TH sur une rétine circulaire de dimension finie, en étudiant la matrice d'accumulation obtenue à partir d'une distribution uniforme de points sur toute l'image.

Cette matrice d'accumulation présente des particularités de distribution qui s'expliquent par le rapport entre le nombre de droites qui passent à proximité du centre de l'image, comparées à celles qui coupent les frontières de la rétine finie. Les premières sont nombreuses et longues, et occupent les valeurs basses de ρ; par contre, les dernières, qui occupent les valeurs élevées de ρ, sont de taille plus réduite, et donc la valeur de leur compteur dans la matrice d'accumulation est plus faible.

De ceci, [COHE77] conclut qu'une TH ne peut être indépendante de ρ, et conseille de recourir à une quantification non linéaire du paramètre ρ qui donne à chaque compteur de la matrice d'accumulation une valeur identique (moyenne) lors de la TH d'une image aléatoire.

[ALAG81] remarque qu'une quantification non uniforme du seul paramètre ρ produit une distribution uniforme des compteurs pour θ fixé. La bonne variation des paramètres dans un espace à 2 dimension est basée sur $(d\rho, d\theta)$, puisque c'est l'invariant le plus naturel qui produise une distribution uniforme des compteurs à partir d'une image à distribution uniforme. La *beta* distribution permet d'obtenir une bonne quantification de l'espace des paramètres.

Plus récemment, l'influence des paramètres a été observée pour des images non pas aléatoires, mais réelles et contenant une certaine forme. Bien évidemment, les lignes ont servi de cobayes. [VEEN81] étudie la détection des lignes avec les paramètres (ρ,θ) et conclut que la forme et la largeur des pics produits par ces lignes dépendent de la quantification de l'espace image et de celle de l'espace des paramètres, ainsi que de la largeur des lignes. Les conclusions de cette étude portent sur la modification des formes des pics, qu'on désire beaucoup plus *aiguës*, en insérant une information supplémentaire qui viendrait pondérer les valeurs accumulées dans la matrice; en l'occurrence il s'agit ici du gradient aux points analysés.

[BALL82] parle de la TH comme d'une méthode efficace d'implantation d'un filtrage par échantillon, i.e. *template-matching paradigm*. On retrouve de fait dans [BROW83] l'idée de filtres spécifiques. La méthode consiste à étudier la distribution des compteurs de la matrice d'accumulation obtenue par la TH d'un objet seul. A cette matrice est associée la notion de fonction, qui projette l'ensemble des points image sur un pic central entouré d'une distribution en lobe, dans l'espace des paramètres. Ainsi la fonction obtenue décrit parfaitement l'objet, et son application à une image contenant l'objet est une *autoconvolution*.

Dans le cas des lignes, ces fonctions s'apparentent plus à des arêtes étendues qu'à des pics étroits. Aussi [BROW83] introduit la TH *complémentaire*, où les points image contribuent à la fois à l'incrémentation de la matrice d'accumulation, mais aussi à la décrémentation de valeurs de cette matrice. Ainsi la recherche de plusieurs objets dans une même image devient possible sans avoir une erreur trop impor-

tante d'existence de droites. Les interférences entre les distributions distinctes dues aux objets peuvent en effet être relativisées dans la matrice d'accumulation. Cependant, si cette méthode permet d'obtenir des fonctions avec des pics proéminents et une base plus étroite, elle apparaît plus sensible à la quantification de l'espace des paramètres.

Approche statistique

Pour clore ce sujet, nous présentons brièvement les idées récentes de [HUNT88] qui établit l'existence d'un lien entre une théorie statistique de la détection de courbes et la TH, qui n'en serait qu'une approximation. Les auteurs rappellent les inconvénients de la transformée de Hough:

(i) La TH favorise les lignes contenant un nombre important de pixels; ce problème s'accroit avec le niveau de bruit dans l'image.

(ii) La TH suppose une distribution *a priori* uniforme des lignes dans l'image. Ceci n'est pas forcément respecté; certaines images contiennent un nombre important de lignes orientées dans la même direction.

(iii) La TH ne prend pas en compte la fonction de densité probabiliste d'un bruit aléatoire dans l'image.

Ils cadrent le problème de la détection d'une ligne dans la théorie du signal, notamment en se référant au critère de Neyman-Pearson pour l'aspect statistique de la décision. En effet, le problème de la TH peut s'énoncer en ces termes:

soit H_0, l'hypothèse $X = B$

soit H_1, l'hypothèse $X = S(\rho,\theta) + B$ pour un (ρ,θ) donné,

où X et B sont des tableaux 2D qui représentent l'un les pixels d'une image, l'autre le bruit aléatoire contenu dans l'image, et $S(\rho,\theta)$ est une image 2D qui ne contient qu'une ligne donnée par les paramètres (ρ,θ) inconnus. Alors, l'image X peut soit ne contenir aucune ligne, il s'agit de l'hypothèse H_0, soit contenir une ligne avec des coefficients inconnus, il s'agit de l'hypothèse H_1; le bruit est toujours présent dans l'image.

On a donc un vecteur d'observation X, et la loi de probabilité qu'on peut introduire dépend de la situation:

- soit il y a absence de signal: hypothèse H_0

  la loi de probabilité = $P(X/H_0)$

- soit une ligne existe, noyée dans le bruit: hypothèse H_1

  la loi de probabilité = $P(X/H_1)$

Pour prendre des décisions, ici elles concernent l'existence ou non d'une ligne, il faut une règle ou fonction de décision qui nous permette d'estimer quelle est l'hypothèse réalisée. Si on nomme δ_0 l'observation de l'hypothèse H_0, et δ_1 celle de H_1, cette fonction donne une estimation des probabilités

suivantes:

hypothèse vraie	observation	appellation	proba
H_0 vraie	δ_0 décidée	non détection vraie	$P(\delta_0/H_0) = \alpha'$
H_0 vraie	δ_1 décidée	fausse alarme	$P(\delta_1/H_0) = \alpha$
H_1 vraie	δ_0 décidée	non détection fausse	$P(\delta_0/H_1) = \beta$
H_1 vraie	δ_1 décidée	détection vraie	$P(\delta_1/H_1) = \beta'$

sachant que $\alpha' = 1 - \alpha$, et $\beta' = 1 - \beta$.

Pour prendre cette décision, il existe en fait deux règles qui dépendent du critère d'optimalisation choisi. Le premier critère est celui de la minimalisation du risque moyen, appelé solution de Bayes. Elle n'est envisageable qu'en connaissance des probabilités de réalisations *a priori* des hypothèses H_1 et H_0, ce qui semble difficile à évaluer.

L'autre critère est celui de Neyman-Pearson, qui maximalise la probabilité de détection vraie β', à probabilité de fausse alarme α donnée. Intuitivement ce critère est plus proche de la réalité. On montre dans ce cas que la règle de décision revient à comparer le rapport de vraisemblance

$$L(X) = \frac{P(X|H_1)}{P(X|H_0)}$$

à un seuil qui est fixé *a priori* en fonction du domaine où la réalisation X implique la décision δ_1.

Nous laissons au lecteur curieux le loisir d'approfondir le développement statistique de $L(X)$ dans [HUNT88, pp.224-225]. La démarche originale qui consiste à retrouver l'algorithme de la TH en dérivant un problème de théorie de la décision, se dégage comme un dessein des recherches entreprises pour améliorer l'efficacité de cette TH. Les outils statistiques puissants qu'offre cette théorie permettent d'exploiter finement des stratégies de choix: ajustement des paramètres en fonction des longueurs des courbes à détecter, de la distribution du bruit, et aussi de la distribution *a priori* des courbes. Ainsi l'estimation des performances se fait à l'aide des COR, courbes des caractéristiques opérationnelles du récepteur qui évalue β' en fonction de α et du rapport signal-bruit à l'entrée.

Considérant un bruit gaussien indépendant spatialement, [HUNT88] obtient une expression de $L(X)$[1]

1. nous ne résistons pas à la tentation de donner l'expression de $L(X)$:

$$L(X) = \sum_{\rho}\sum_{\theta} exp\left(\frac{1}{\sigma_n^2}\left(\sum_{(i,j)\in T_{\rho,\theta}} X_{ij}S_{ij}(\rho,\theta) - \frac{1}{2}\sum_{(i,j)\in T_{\rho,\theta}} S_{ij}^2(\rho,\theta)\right)\right) P(\rho,\theta|H_1)$$

où i et j indexent les lignes et colonnes de X, et $T_{\rho,\theta}$ est un sous-ensemble des pixels présents dans $S(\rho,\theta)$. La partie sur fond pointillé correspond au contenu d'un compteur de la matrice d'accumulation dans la TH. On lui soustrait une valeur relative à la longueur de cette ligne $\sum S^2(..)$. Chacun de ces termes est ajusté et élevé de façon exponentielle. La dernière étape consiste à moduler l'expression obtenue par la distribution de probabilité *a priori* des lignes dans l'image. Ceci met en jeu les trois désavantages cités de la TH, et donne ainsi un critère optimal de décision sur l'existence ou non d'une ligne dans l'image.

qui se décompose en un terme correspondant à la valeur calculée en chaque point de la matrice d'accumulation de la TH, à laquelle on soustrait une valeur correspondant à la longueur de la ligne. Ainsi cette expression de *L(X)* montre clairement que la TH n'est qu'une version grossière d'un calcul plus complexe et précis, et qui concerne l'estimation d'une décision optimale sur la présence ou non d'une ligne dans l'image.

Cet article présente donc une méthode qui s'avère plus puissante que la TH si les caractéristiques du bruit sont connues d'avance. Dans le cas contraire, la TH est une bonne approche d'une méthode qui présente le désavantage d'être beaucoup plus lourde à mettre en oeuvre et nécessite beaucoup plus de calcul que la simple TH. On peut se réjouir de ce qu'une méthode intuitive telle que la TH procède de l'approximation *a posteriori* d'une technique basée sur un raisonnement statistique complet et démontrable.

2.2.4 Approche hiérarchique de la transformée de Hough

Les méthodes d'approximation polygonale se basent sur la *proximité* des pixels, en parcourant les pixels par voisinage et en essayant de décrire cette suite à l'aide d'un segment tant que l'erreur d'approximation n'est pas trop élevée. La TH détecte les pixels *colinéaires*, par la méthode décrite. [PRIN90] tente de réunir ces deux contraintes, proximité et colinéarité, dans une approche hiérarchique de la TH.

Cette approche se base sur une première description en segments de petite taille, qui est obtenue par application de la TH sur des sous-images se recouvrant (*overlapping subimages*). Ensuite le processus se hiérarchise de façon verticale. Les petits segments d'un voisinage sont regroupés en segments plus importants, toujours par application d'un algorithme du type TH. Ainsi des segments d'un niveau qui se trouvent localement groupés, peuvent générer un segment de taille plus importante au niveau supérieur. La description finale est hiérarchisée, et la longueur d'une ligne détermine les niveaux jusqu'auxquels celle-ci s'est propagée.

Sur la taille de la rétine

L'approche hiérarchique fait directement référence aux risques de détection fausse dont nous avons déjà parlé, risques relatifs au choix de certains paramètres. Fort des analyses effectuées sur l'efficacité de la TH appliquée à une rétine de dimension finie, [PRIN90] introduit comme nouveau paramètre la taille de cette rétine dans la phase de regroupement des pixels que génère la TH.

En effet, plus la taille de cette rétine augmente, plus les risques de trouver une *pseudo*-colinéarité de pixels, due au bruit, augmentent. A l'opposé, pour une rétine de taille réduite, l'interpolation discrète des lignes contribue à la diminution du nombre d'orientations possibles de ces lignes dans cet espace. Ainsi la réduction de la taille de la rétine améliore les chances de détection vraie d'une ligne, et diminue la complexité du problème en influant directement sur le nombre de valeurs de θ distinctes pour cette TH.

Tout ceci motive le recours à une approche hiérarchique. La notion de voisinage permet une meilleure détection des groupes de pixels colinéaires. L'aspect hiérarchique de la transformation facilite la variabilité ascendante des paramètres, pour obtenir des critères de succès de plus en plus fins. Enfin il introduit la notion de proximité de façon transparente.

L'algorithme

Le paragraphe *3* de [PRIN90] donne une description précise et détaillée de l'algorithme utilisé, qui ne présente aucune originalité quant au calcul de la TH. Aussi il ne nous semble pas utile de le rappeler ici, mais bien plus d'insister sur quelques points particuliers qui rejoignent notre démarche.

La structure utilisée est une pyramide classique de $N+1$ niveaux. A chaque niveau n de cette pyramide, l'image est divisée en $2^{N-n}x2^{N-n}$ sous-images avec $n=0$ et $n=N$ pour le niveau le plus bas et le plus haut respectivement. La méthode de filiation emploie le mécanisme de recouvrement suivant: un parent du niveau $n+1$ veille sur un voisinage de $4x4$ sous-images du niveau n, dont le groupe central de $2x2$ sous-images est son fils (cf. figure 5). Cette technique de recouvrement pyramidal se retrouve chez [SHNE81].

Au niveau le plus bas, la matrice d'accumulation est réduite par la taille étroite de la rétine. Par contre aux niveaux plus hauts, cette matrice peut s'agrandir pour rendre plus précise la description des lignes reconnues.

A chaque niveau de la pyramide, un nouveau groupe de pixels colinéaires peut se former quand on retrouve des pics pour un même couple de paramètres dans les matrices d'accumulation de sous-images voisines du niveau précédent. Si un groupe ne peut plus se propager faute de pics dans les matrices voisines, le niveau à partir duquel ce groupe cesse sa propagation détermine la longueur de la ligne détectée.

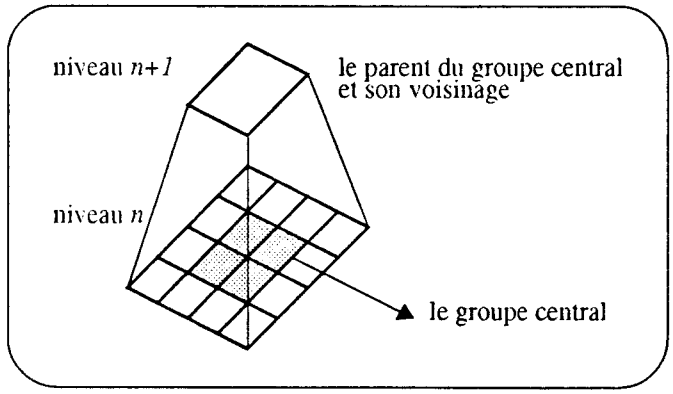


figure 5 Voisinage d'un parent pour former un regroupement de segments

Au niveau le plus bas, et sur une image *256x256* pixels, les sous-images ont une taille de *16x16* pixels, qui nécessite donc une matrice d'accumulation *16x16*. La procédure d'accumulation se répète jusqu'à obtenir une valeur *8* fixée de détection d'un pic. Ensuite [PRIN90] donne l'illustration de cette première phase de détection en recréant des petits segments, pour les paramètres ayant généré des pics. La variation du paramètre θ à ce niveau se fait donc par incrément de $\pi/16$. Les temps de simulation sur Micro VAX II se situent entre 3/4 d'heure et une heure, ce qui donne une idée de la complexité d'un tel algorithme sur une machine séquentielle.

De cet article, nous retiendrons donc que l'approche hiérarchisée introduit l'aspect de *proximité* des pixels dans la TH, sans ajout d'un autre paramètre. Cette proximité améliore la détection vraie des lignes; elle diminue la taille mémoire requise localement pour la matrice d'accumulation, ce qui facilite l'implantation de l'algorithme sur une machine parallèle dotée de peu de mémoire locale par PE. De plus, elle génère artificiellement une approximation de cette détection sur le paramètre θ, quand on se place à un niveau de la pyramide. Nous verrons que cet effet rejoint notre approche, et participe étroitement à un paradigme *d'approximation* dans les mécanismes de la vision.

2.2.5 *Architecture et Transformée de Hough*

Le lecteur trouvera une description détaillée des applications de la TH dans [ILLI88, pp.103-107]. Nous avons au long de l'exposé fait allusion à la complexité de la TH, notamment quant au nombre de paramètres utilisés pour décrire la courbe recherchée. Cette complexité croissante restreint souvent l'usage de la TH à la recherche de lignes [LEVI85].

La TH consiste essentiellement en un certain nombre d'opérations très simples effectuées indépendamment. La difficulté provient du partage des données, que ce soit pour les entrées *(x,y)* ou pour les sorties[1] (ρ,θ). Nous présentons ici les implantations les plus cohérentes avec l'aspect parallèle de l'algorithme, que nous pensons être les implantations soit SIMD soit systolique. La raison de ce choix vient du caractère global du traitement, que naturellement nous imaginons comme projeté sur une machine à *n* processeurs qui feraient en même temps les calculs simples de transformation de l'espace image à l'espace des paramètres. Le choix MIMD présente un réel désavantage au niveau des communications, dont le taux est élevé [OLSO87].

Approche SIMD

[ROSE88] présente l'utilisation de la TH pour le projet de véhicule autonome, financé par le DARPA. La TH est implantée sur deux machines SIMD que nous avons présentées: MPP et le GAPP.

Soit une image de taille *nxn*, alors la valeur limite de ρ est $n\sqrt{2}$, donc du même ordre que *n*. Le nombre de lignes discrètes qui passent par un point donné dans une image *nxn* est aussi de l'ordre de *n*, ce qui signifie que ces lignes ont *O(n)* pentes différentes. Ainsi l'espace des paramètres pour la TH est de taille *O(nxn)*. Pour cette raison, les applications de la TH sur des machines SIMD s'effectuent avec une machine à *nxn* processeurs élémentaires, pour transformer une image de *nxn* pixels dans une matrice d'accumulation (ρ,θ) de taille *nxn*.

[SILB85] décrit un algorithme de TH pour une machine SIMD, dans lequel cette transformée est calculée en parallèle pour une valeur de θ à la fois. Chaque PE contient un pixel, et les PEs dont le pixel est noir connaissent leur valeur de θ, obtenue par le gradient de Sobel (cf. paragraphe *2.3.5* à la page 1-18). De plus, chaque PE stocke le compteur de coordonnées (ρ,θ) de la matrice d'accumulation.

L'algorithme est donc le suivant:

Pour toutes les valeurs de θ

Envoyer la valeur θ, $cos\theta$ *et* $sin\theta$ aux PEs,

Chaque PE soustrait la valeur de θ reçue à la sienne

Si $|\theta_{reçue} - \theta_{propre}| > \theta_{référence}$

alors bit de validité du PE = 0

sinon le PE calcule $\rho = x.cos\theta + y.sin\theta$,

où *(x,y)* sont les coord. du PE

Mise à jour de la matrice d'accumulation

finpour

La procédure de mise à jour de la matrice d'accumulation évalue le nombre de PEs qui possèdent la

1. La plupart des études architecturales et des implantations de la TH utilisent les coordonnées polaires.

même valeur ρ après leurs calculs. Puisqu'on travaille pour un θ donné, ce nombre est la valeur qui doit être stockée dans la matrice en (ρ,θ), c'est à dire dans le compteur du PE de coordonnées (ρ,θ). La procédure se déroule comme suit:

Décaler verticalement *n fois* les valeurs ρ, Pour chaque décalage:

(les PEs des bords sont rebouclés en cycle)

Si ($\rho_{reçue} = x_{PE}$)

alors *ADD 1* dans un registre temporaire de ce PE

Décaler horizontalement *n fois* les reg.temp., Pour chaque décalage:

(les PEs des bords sont rebouclés en cycle)

Si (y_{PE} = *valeur* θ *traitée*)

alors *ADD la valeur du reg.temp.* dans le compteur final du PE(ρ,θ)

[ROSE88] donne une description détaillée du nombre de cycles machine requis par cet algorithme complet. Si le nombre d'angles pour lequel se déroule l'algorithme est de l'ordre de n, alors le temps est de $O(n^2)$ cycles machine.

Plus précisément, puisque la machine utilise des PEs 1 bit, et que des additions[1] sont effectués avec une précision en $log_2(n)$ bits, le temps est en $O(log_2(n).n^2)$ cycles machine. Il est important de préciser que le facteur $log_2(n)$ n'intervient pas du fait des niveaux de gris; l'image est d'abord traitée par le gradient de Sobel, et n'est composée que de pixels blancs ou noirs.

Sur la complexité du traitement

Enfin, nous remarquons que cette valeur $O(log_2(n).n^2)$ tient à trois cotés de l'algorithme: d'une part le fait qu'on utilise des calculs arithmétiques dans la TH, auxquels correspond le terme $O(log_2(n))$, d'autre part le fait d'effectuer cette TH avec $O(n)$ angles θ distincts, et enfin les $O(n)$ valeurs distinctes du paramètre discret ρ. Si nous situons la démarche qui fût la nôtre par rapport aux modifications possibles de cette valeur précisément définie et grande[2], il convient de déterminer la cause effective des différents rapports et de réfléchir sur l'inéluctabilité de cette complexité.

Nous remarquons en effet que la TH nécessite l'utilisation d'opérateurs arithmétiques (addition et multiplication) qui requièrent sur une machine SIMD 1 bit de l'ordre de $log_2(n)$ opérations 1 bit, si les valeurs traitées appartiennent à $[0,n]$. Le recours à une machine SIMD $log_2(n)$ bits permettrait d'effectuer ces opérations en un temps de l'ordre d'*1* cycle machine. Cependant, les contraintes sont plus complexes, et les degrés de liberté moins évidents qu'on ne pourrait l'imaginer de prime abord. En effet, le choix d'une largeur plus importante pour le PE diminue les niveaux d'intégration de ces PEs sur un CI, et donc de réalisation totale de la machine. Deux paramètres sont à prendre en compte: d'une part la taille du PE par lui-même (la taille de l'UAL n'est pas forcément l'élément majeur, contrairement à la

---

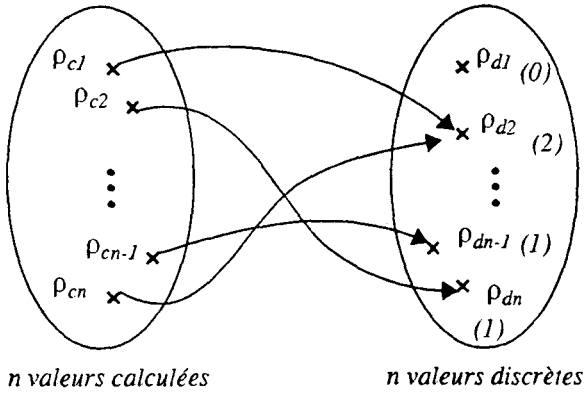
1. Les multiplications nécessitées par la TH sont écartées dans [ROSE88]; peut-être cela est-il dû à l'espace discret des paramètres (ρ,θ), qui permet un calcul approché du résultat, et donc requiert moins d'additions. Néanmoins, nous ne sommes pas complètement satisfait par cette explication: toute multiplication est plus longue qu'une addition, et donc réduit *ipso facto* l'efficacité de l'algorithme. Il nous paraît extraordinaire de trouver dans un article aussi bien détaillé dans son ensemble, une telle zone d'ombre...

2. Comme nous l'avons déjà indiqué au premier chapitre, les valeurs de n, telles $n=128$ ou $n=512$, sont bien trop petites comparées à ce qu'un satellite fournit comme image, ou à ce que l'oeil humain voit (n est dans ces deux cas plus proche de *10000* que de *128*...).

taille des mécanismes de sélection tels que, par exemple, le sens du transfert I/O entre les PEs voisins), d'autre part les chemins de données entre PEs qui, bien qu'en réseau maillé, peuvent encombrer le circuit.

Si, malgré ces contraintes, l'architecte réalise une machine SIMD $log_2(n)$ bits, la taille d'un PE induira une diminution de la vitesse à laquelle le circuit fonctionne, quelles que soient les technologies employées. La raison à ceci vient de la rapidité des opérateurs logiques comparés aux opérateurs arithmétiques, simplement parce qu'une addition comporte plusieurs portes logiques. Quoique tout traitement arithmétique n'est pas son pendant logique, la plupart des machines SIMD sont des machines 1 bit. Le lecteur intéressé par ces questions insolubles pourra se reporter aux commentaires prolixes de [HILL85] sur le choix de la taille d'un PE de la Connection Machine.

Pour un angle θ donné, $O(n)$ opérations sont effectuées pour évaluer le nombre d'occurrences d'une valeur de ρ dans chacune des colonnes du réseau 2D de PEs. L'opération effectuée est une application de l'ensemble des n valeurs de ρ calculées sur l'ensemble des n valeurs de ρ discrètes qui composent l'espace des paramètres; suivie d'une évaluation pour chaque élément de l'ensemble d'arrivée du nombre d'éléments de l'ensemble de départ qui s'y appliquent. Une telle opération est similaire à un tri[1];



elle ne peut donc se faire *au mieux* qu'en un temps $O(log_k(n))$. Par contre la dernière phase de l'algorithme, c'est à dire le rangement de la valeur de ρ calculée pour notre angle θ, n'est qu'un stockage de n données indépendantes dans n cases distinctes de la matrice d'accumulation. Ce stockage prend dans notre cas n étapes, du fait de l'implantation choisie de cette matrice (*1 case = 1 PE*). Il n'est pas évident que pour tout autre méthode d'implantation, cette valeur $O(n)$ résiste à l'ingéniosité des architectes...

Ce qui peut être retenu sur la complexité de l'algorithme de Hough et son implantation sur une machine parallèle, se résume en trois points:

(i) le calcul des valeurs (ρ,θ) à partir des coordonnées (x,y) des pixels noirs doit s'effectuer à θ donné, pour éviter l'explosion du nombre de couples obtenus dont on ne saurait quoi faire; la vitesse des UAL est ici critique;

(ii) une fois calculée les valeurs de ρ à θ donné, l'évaluation du nombre d'occurrence d'un ρ donné peut être assimilée à un tri;

1. La méthode d'obtention du vecteur (pour un angle θ donné) de la matrice d'accumulation est la suivante:
(1) on trie les n valeurs calculées ρ_{ci}, ce qui nous donne une liste de valeurs successives ρ_{icj};
(2) pour chaque couple $(\rho_{icj}, \rho_{icj+1})$ tel que $\rho_{icj} \neq \rho_{icj+1}$, on fournit les index j et $j+1$ à l'étape
(3) qui évalue ainsi le nombre d'occurrence de ρ_{dk}
(par exemple: $\{\rho_{icj} = \rho_{icj+1} = ... = \rho_{icl}\} = \rho_{dk}$ donc $occur(\rho_{dk}) = l - j$).
On voit que le tri est la partie la plus importante de l'algorithme, le reste peut se faire rapidement (en temps constant pour peu qu'on utilise une architecture adaptée).

(iii) le stockage de la matrice d'accumulation peut ne pas trop limiter l'efficacité de l'algorithme, pour peu qu'il existe une disposition cohérente avec les étapes précédentes et à accès parallèle.

Le lecteur désireux d'approfondir cette réflexion, dont l'intérêt croît relativement à la valeur de n, peut se rapporter au paragraphe *4.3* de [ROSE88], où il trouvera une description d'une technique de tri des valeurs (ρ,θ), intéressant dans le cas où n devient grand.

Pour situer ce problème de complexité, [ROSE88] donne le temps de calcul de la TH sur la machine MPP, avec l'algorithme présenté précédemment: **130 ms**. Nous rappelons que MPP possède une matrice de 128x128 PEs avec un temps de cycle de 100 ns. Aussi nous laissons au lecteur le soin d'imaginer le temps requis pour traiter une image 1000x1000, même s'il existait une machine de 10^6 PEs au temps de cycle de 10 ns par exemple....

De l'idée de projeter

L'algorithme présenté est profondément inspiré de celui de Silberberg, dont on connaît bien le comportement en $O(N.p)$, où N est le nombre de pixels de contour traités et p le nombre de valeurs de θ. Cypher a récemment présenté un algorithme en $O(N+p)$ pour une machine SIMD tableau de NxN PEs, avec projection sur chacun des PEs d'un pixel de l'image. [PAN90] propose un algorithme similaire, masi fonctionnant pour les pixels contour uniquement (on ne traite que les pixels extraits comme appartenant à un contour). Pour N^2 points de contours, chacun des PEs de la machine tableau reçoit les coordonnées d'un de ces points de contour. De plus, sur chaque PE sont réservés un tableau de p compteurs et une variable. Dans cette variable, et pour tous les PEs de la ligne i[1], on charge la valeur θ_i.

L'algorithme consiste alors à translater les coordonnées des pixels dans les colonnes, afin que ceux-ci rencontrent toutes les valeurs possibles de θ. Pour chacune de ces valeurs, le scalaire ρ correspondant est calculé à l'aide des coordonnées (x,y) du pixel. Le PE incrémente alors la $\rho^{\text{ième}}$ valeur de son tableau de p compteurs. Ceci terminé, en $O(p)$ cycles, chaque tableau d'une même ligne correspond à un même angle θ donné. Pour reformer la matrice d'accumulation, il suffit de faire circuler les valeurs de cette matrice d'accumulation, $M(\theta_i, \rho_j)$ pour j=0..p à la ligne i. Ceci s'obtient en $O(N)$ cycles. L'algorithme obtenu est donc en $O(N+p)$.

Architecture systolique

[DEUT89] présente succinctement l'implantation d'une TH sur une machine à base de processeurs WARP. L'image traitée possède *512x512* pixels 1 bit, et la matrice d'accumulation *180x512* éléments. Pour chaque pixel *(x,y)* de valeur 1, et chaque *i*, *0<i<180*, on incrémente l'élément *(i,j)* de la matrice d'accumulation où *j* est donc la distance perpendiculaire du point *(0,0)* à la ligne passant par *(x,y)* qui fait un angle *i*-degrés avec l'axe *Ox*.

Les couples *(x,y)* traversent la ligne de processeurs (72 processeurs WARP sur la machine iWarp), et chaque processeur incrémente la partie de la matrice d'accumulation qu'il gère. Sur cette machine iWarp, 60 processeurs sont réellement utilisés qui prennent chacun en charge le calcul pour 3 angles *(3x60=180 degrés)*, et stockent donc chacun *3x512* éléments de cette matrice d'accumulation. Le temps d'exécution estimé de la TH est de **60 ms** sur cette machine qui offre tout de même une puissance de

1. Concernant cette projection des N^2 pixels de contour sur une machine à NxN PEs, nous n'avons pas trouvé une explication précise du cas où le nombre p de valeurs possibles de θ serait plus important que N. A notre avis, la raison en est qu'il faut utiliser un réseau de Nxp PEs dans ce cas. Trivialité que les auteurs auront escamotée.

1,15 GFlops! Implantée sur PC Warp, qui ne possède que 10 processeurs WARP et n'offre donc qu'une puissance de 100 MFlops, la TH prend **340 ms**.

Pour relativiser ces résultats, il faut noter que l'algorithme utilisé croît linéairement en fonction du nombre de pixels noirs (bit = 1) dans l'image, et que ces résultats sont obtenus ou estimés pour une image supposée contenir *10%* de pixels noirs...

Processeurs dédiés

[LAHA86] propose une méthode associative de calcul de la TH. Pour un pixel donné (x,y), le calcul effectué est l'association à chaque valeur de a_i de l'échantillon b_i le plus proche, suivant le critère:

$$b_i - db/2 < y - a_i.x < b_i + db/2$$

La valeur de b_i est vue comme le résultat de l'application d'une fonction simple sur a_i. Ceci permet de: *(i)* déterminer les $b_i(a_i)$ de façon indépendante, en parallèle, et *(ii)* de calculer cette valeur $b_i(a_i)$ en un temps indépendant de la valeur a_i.

Ces propriétés permettent de définir une architecture parallèle et câblée. Celle-ci se compose de modules identiques effectuant le même calcul, à chacun desquels est associée l'une des valeurs a_i de l'espace des paramètres de la TH. Ce calcul peut se faire de manière synchrone, et d'autant plus rapidement que les calculs sont simples. On retrouve dans cette intégration un fonctionnement identique à celui des algorithmes pour machines SIMD.

2.2.6 Conclusion

Nous n'avons pas abordé la question de la recherche des pics dans la matrice d'accumulation. La méthode la plus courante est évidemment une recherche par détermination d'un seuil global [ILLI88]. De même que la taille de la matrice d'accumulation, et donc le nombre de calculs, cette recherche dépendra étroitement de la dimension de l'espace des paramètres.

En guise de conclusion, nous remarquons simplement que cette méthode demande une puissance de calculs importante: nous avons parlé de la machine *iWarp* à 1,15 Gflops, ce qui n'est pas rien... Aussi, et puisque cette transformée excite l'intérêt des concepteurs d'un véhicule autonome, mais aussi certainement celui des concepteurs de systèmes de suivi et verrouillage de cibles, nous laisserons au lecteur la possibilité d'entr'apercevoir un lien entre notre proposition et la TH. Comme le suggère [LEVI85], une analogie fonctionnelle existe entre cette TH et une aire visuelle de notre cortex. Cette aire n'a pas, selon toute vraisemblance, formé une représentation abstraite de l'image, qui serait calculée paramétriquement. Aussi nous sommes-nous orientés vers une démarche cellulaire.

2.3 Convolutions et Filtrages

2.3.1 Introduction

Sous un titre mal choisi, mais en est-il de meilleur, on s'intéresse ici aux techniques transformant l'image elle-même, sans la transposer dans un espace abstrait comme le proposent les techniques précédentes qui se basent sur les coordonnées des pixels noirs de l'image. L'intérêt d'une telle démarche provient de ce qu'une comparaison logique est beaucoup plus rapide qu'un calcul arithmétique. L'application de ceci à la reconnaissance des formes se traduit par le *matching*, terme difficilement transposable en français, qui signifie qu'on tente d'*égaler* des formes, de retrouver celles-ci dans l'image, sans passer par une description "abstraite" de ces formes.

Nous avons vu combien la transformation de Hough est pénalisée par les calculs effectués. Les méthodes que nous présentons ici cherchent à se défaire de ce manque d'efficacité, en comptant sur la rapidité des calculs purement logiques. Ainsi, des méthodes qui ne participent pas de descriptions paramétriques, permettent d'obtenir des opérateurs réellement *temps réel au sens de la vision*, comme on peut en trouver en morphologie mathématique.

Toutefois il n'est pas dans nos intentions de ramener les découvertes dans ce domaine, important dans le traitement d'images, à une simple expression d'un besoin d'opérateurs rapides. Ce ne serait d'ailleurs pas le cas pour le *template matching* qui appartient à la famille des convolutions, et utilise les outils arithmétiques de manière intense. Cependant, si l'on veut élargir sa propre vision, on est alors amené à considérer ces méthodes, complémentaires aux précédentes, dans le cadre informationnel de l'accès à des données. En effet, chacun sait qu'il existe intrinsèquement deux types d'accès à toute information: soit par index, soit par associativité. La géométrie analytique est un bon moyen d'obtenir un index (ici des paramètres) pour rechercher les modèles similaires, à partir de la figure analysée. Par complémentarité, la géométrie élémentaire accède de manière associative[1] (en comparant le contenu) à une description de cette figure en formes de base connues. Le terme *filtre* est alors employé à bon escient.

De ces techniques de filtrage spatial, on obtient des informations sur l'existence et leur position de formes recherchées[2] dans l'image. Pour une extraction de segments, ces méthodes sont particulièrement intéressantes puisqu'elles permettent d'extraire de petits segments rapidement dans l'image. Les critères de recherche (convolution, érosion, dilatation) précisent la qualité de l'information: l'érosion étant le meilleur critère.

Cependant, ces filtres extraient une information pauvre qui doit être analysée plus finement. Ainsi l'extraction par filtrage de petits segments ne donne pas la liste des segments contenus dans l'image. Il n'est pas raisonnable de chercher à extraire directement un segment de grande longueur, car cela nécessiterait de gros moyens *hardware* (ou du temps) et devrait être fait pour tous les segments possibles d'une image... Les techniques analytiques peuvent donc intervenir à ce niveau, pour décrire une suite spatiale

1. Il est remarquable que les techniques connexionnistes, qui sont des techniques associatives, s'apparentent aux techniques de filtrage et plus généralement à la théorie du signal dans cet aspect associatif de l'accès à l'information. Toutefois les techniques connexionnistes proposent une analyse brute du contenu de l'image: il existe ou non une forme connue par le système dans l'image. Les techniques de *matching* filtrent l'image pour renforcer les formes qui *ressemblent* à celles déterminant le (ou spécifiques du) filtre. On ne fait pas exactement de l'analyse, mais du traitement d'images.

2. Nous appellerons modèle de filtre, une forme recherchée par une technique de filtrage spatial. En Vision, les modèles utilisés seront donc les différentes formes élémentaires, primitives, qui servent de base à la définition de formes plus complexes. Les primitives choisies pour Pastis sont le segment, l'arc de cercle; viennent ensuite le carré, le triangle, le rond,... En théorie du signal, le modèle d'une transformée de Fourier serait la fonction sinusoïdale complexe: $e^{2i\pi tx}$.

de petits segments sous la forme d'une ou plusieurs équations de droites.

Cependant, les modèles des filtres peuvent, dans certains cas, contenir une information de caractère *quasi*-analytique. La recherche d'un segment de taille importante dans une certaine direction de l'image contient une définition analytique de l'objet cherché (ρ,θ). Cette information est certes proche de la transformée de Hough; elle reste à exploiter.

2.3.2 *Le Template Matching*

Le Template Matching fait partie des processus les plus bas en traitement d'images. Il s'agit en effet d'une technique de filtrage, qui partant d'un modèle recherché (*template*) tente de le retrouver dans l'image par une convolution spatiale[1]. Ce modèle est en fait une sous-image qui contient complètement l'image de l'objet recherché. Une mesure de ressemblance est calculée qui indique la distance entre l'image analysée et l'objet recherché, en tout point de celle-ci. Le point où cette valeur est maximale, peut être sélectionné comme localisant l'objet dans l'image.

Une mesure standard entre une fonction $f(x)$ et un modèle $t(x)$ est la distance euclidienne $d(x)$, [BALL82], définie par:

$$d^2(x) = \sum_{y=-M}^{M} (f(x+y) - t(y))^2$$

Si l'image correspond exactement au modèle au point x, alors $d^2(x) = 0$; sinon $d^2(x) > 0$. En développant l'expression ci dessus, on obtient:

$$d^2(x) = \sum_{y=-M}^{M} (f^2(x+y) - 2f(x+y)\,t(y) + t^2(y))$$

La somme des carrés $t^2(y)$ est une constante qui peut être ignorée. De même, quand la somme des $f^2(x+y)$ est presque constante, alors le produit $f(x+y)t(y)$ indique la distance entre $f(y)$ et $t(y)$ au point x. Ce produit est appelé produit de convolution, il se calcule de la manière suivante: soit I[0..N-1,0..N-1] une image NxN et T[0..M-1,0..M-1] un modèle de dimension MxM, où M est généralement petit devant

1. Le template matching et la transformée de Fourier, souvent utilisée pour filtrer l'image, ont en commun qu'il s'agit de produits de convolution: le premier est spatial, le second est fréquentiel. La transformée de Fourier Discrète, TFD, est la version "spatiale" de la transformée de Fourier générale. L'idée essentielle, qui donne naissance à ce type de transformations, vient des résultats obtenus dans les espaces vectoriels normés complets: les espaces de Lebesgue. Les théorèmes de densité dans ces espaces démontrent que toute fonction est limite, au sens de la norme de l'espace, d'une suite de fonctions très simples.

Ces fonctions sont dans le cas de la transformée de Fourier discrète les fonctions sinusoidales à différentes fréquences. L'intérêt de la transformée de Fourier est d'établir un lien entre le groupe des translations et le groupe multiplicatif des complexes de module unitaire, *via* le groupe additif des réels. Dès lors la TFD convertit un produit de convolution en un produit simple: multiplication, et réciproquement. Comme de plus il existe un réseau nommé papillon (*butterfly*) au fonctionnement systolique qui réalise bien cette TFD avec des coûts moindres, ceci explique que la TFD apparaisse comme un outil important soit de filtrage soit de modèle de représentation d'images [BALL82].

N, alors le produit de convolution de I par T donne une matrice K de dimension NxN, telle que:

$$K[i,j] = \sum_{u=0}^{M-1} \sum_{v=0}^{M-1} I[i+u,j+v] \times T[u,v] \qquad pour \qquad 0 \leq i,j < N-M$$

Le *template matching* peut être vu comme le décalage du modèle au travers de l'image; et pour chacun des déplacements, les valeurs superposées du modèle et de l'image sont multipliées, et leurs produits accumulés. On peut aussi calculer la matrice K, en permettant au modèle de parcourir totalement l'image, c'est à dire les points frontières inclus. Dans ce cas, on remplace dans l'équation précédente $I[i+u,j+v]$ par $I[i+u \bmod N, i+v \bmod N]$, ceci pour i et j parcourant 0 à N-1. Cette convolution est nommée convolution périodique, et se retrouve souvent [RANK90].

D'un point de vue plus général, le *template matching* est souvent utilisé en traitement d'images, que ce soit pour la détection de contours ou d'objets, le filtrage,... Il appartient à la famille des convolutions, pour lesquelles existent de bonnes architectures systoliques que nous avons déjà présentées (cf. paragraphe *4.1.3* à la page 1-36). Sa forte complexité en $O(M^2N^2)$ a abouti à des mises en oeuvre le plus souvent SIMD de cet algorithme. Nous en avons donné un exemple au chapitre 1, avec la présentation de l'algorithme du gradient de Sobel, développé sur la machine MPP, et qui n'est autre qu'une convolution 3x3.

Les algorithmes SIMD utilisent l'envoi global en $O(1)$ d'opérandes à tous les PEs, tel qu'il existe sur MPP par exemple, pour obtenir des algorithmes en $O(M^2+logN)$ [KUMA87]. Bien entendu, on suppose que la machine possède N^2 processeurs élémentaires! [RANK90] *s'escrime* à développer un algorithme de convolution sur un hypercube NCUBE de 64 processeurs, i.e: une machine MIMD. L'algorithme est cinq fois plus lent qu'un algorithme optimisé sur CRAY2 mono-processeur (SIMD). Pour donner un ordre de grandeur, avec l'algorithme SIMD, une convolution 4x4 sur une image 64x64 prend 7 ms sur CRAY2; la convolution 4x4 sur image 512x512 prend 273 ms; la convolution 8x8 sur image 512x512, une seconde.

2.3.3 Le filtrage morphologique

Depuis le début de ce chapitre, l'image est décrite comme un ensemble de points ayant un certain niveau de gris[1], les pixels. Sur ce signal, l'ensemble des pixels distribués dans l'espace 2D, les traitements sont linéaires et s'appuient sur la théorie du signal. Que ce soit l'approximation polygonale, qui décrit tout ensemble de pixels à une distance d'erreur d'une forme cherchée. Que ce soit le *template matching*, qui filtre une scène pour en extraire les points dont leur voisinage immédiat ressemble au modèle du filtre.

L'apport des méthodes morphologiques consiste en l'importance accordée aux objets, à leur forme, et non plus à une description abstraite de celles-ci. L'originalité provient de cette séparation de l'analyse numérique des formes, comme décrites par un ensemble de fonctions combinées linéairement, pour aller vers une description ensembliste des images: l'objet est un sous-ensemble de l'espace image, et les combinaisons d'objets deviennent ici une union, là une intersection,...

1. La restriction de notre étude aux images binaires, i.e. formées de points blancs ou noirs, n'empêche pas sa généralisation à toute autre forme de codage. En effet, tout image composée de pixels à niveaux de gris multiples peut être traitée comme plusieurs images binaires: chacune correspond au résultat d'un seuillage d'un niveau de gris particulier.

Définitions

Soit un espace E, les objets de cet espace sont les sous-ensemble X, tel que $X \subset E$. Aussi ce qui nous intéresse c'est l'ensemble des parties de E, $\wp(E)$, qui présente les bonnes propriétés suivantes [SERR86]:

(i) $\wp(E)$ est un treillis complet, qui est muni de la relation d'ordre partiel appelée inclusion, et notée: $\subset$,

(ii) $\wp(E)$ est complet, et donc chaque famille finie ou non d'éléments $X_i \in \wp(E)$ possède un plus petit élément, leur union $\cup X_i$, et un plus grand, leur intersection $\cap X_i$, qui appartiennent tous deux à $\wp(E)$,

(iii) le treillis est distributif, c'est à dire:

$$X \cup (Y \cap Z) = (X \cup Y) \cap (X \cup Z) \qquad \forall X, Y, Z \in \wp(E)$$

et chaque élément $X \in \wp(E)$ possède son complémentaire X^c, tel que:

$$X \cup X^c = E \text{ et } X \cap X^c = \varnothing$$

[SERR86] introduit trois relations possibles entre sous-ensembles:

(i)	B est inclus dans X,	$B \subset X$,
(ii)	B intercepte[1] X,	$B \cap X \neq \varnothing$,
(iii)	B n'intercepte pas X,	$B \cap X = \varnothing$,

dont la figure 6 donne une idée visuelle.

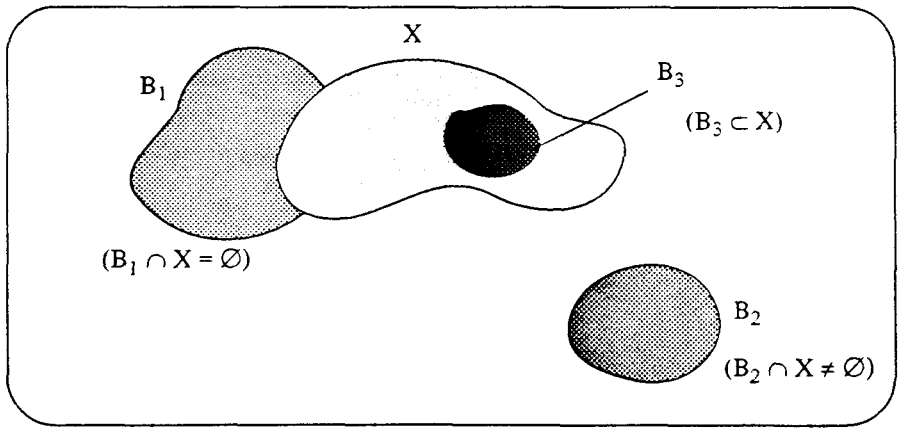


figure 6 relations morphologiques

Le concepteur donne généralement à une machine de vision un objectif fini, qui est la décomposition d'une image en plusieurs objets reconnaissables par l'oeil humain. Cette décomposition n'est cependant pas un paradigme. Nous l'avons tous empiriquement découvert, lors de séances de lecture -hivernale?- de revues hebdomadaires, programmes en tout genre et surtout "grand public", quand avec désespoir

1. Nous insistons sur l'étymologie de ce verbe, provenant du latin *interceptus*, c'est à dire arrêter au passage; de cette façon dirions-nous qu'un nuage intercepte les rayons du soleil, avec l'idée sous-jacente ici du flot de photons capturé, là de l'étendu des niveaux de gris caché.

nous cherchions cette dernière différence qui, parmi les sept annoncées d'avance, représentait l'ultime écart entre deux scènes. Bien que cette scène nous l'eussions comprise rapidement, au pis aller notre oeil s'en fatiguait de n'y trouver plus rien. Ainsi notre vision, trop habitué au travail immédiat de reconnaissance d'objets, se retrouvait bluffé par une scène si simple. L'esprit, se reprenant, décidait alors un crible séquentiel, et l'oeil d'analyser de proche en proche chaque objet, jusqu'à cette septième traîtresse; à moins que de fatigue nous eussions décidé avant sa découverte de mettre fin à ce sacrifice sordide.

Cette mésaventure est le reflet parfait de ce dont nous parlons. Cette analyse séquentielle, lourde mais méthodique, l'analyse morphologique la met en oeuvre. Elle se donne un objet, et parcourt l'image avec celui-ci à la manière d'une convolution. Les opérations ne sont pas arithmétiques, mais ensemblistes: $\subset$, $\subseteq$, $\not\subset$. L'objet est appelé pour l'occasion *élément structurant*, et les tâches sont soit la dilatation, soit l'érosion.

La dilatation de l'objet X par un élément structurant B, est l'ensemble de tous les points x de l'image tels que B(x) intercepte X, où B(x) correspond à l'élément structurant placé de façon que son centre soit x. Critère plus difficile à satisfaire, l'érosion d'un objet X par un élément structurant B est l'ensemble des points x tels que B(x) soit inclus dans X.

$$erosion_B(X) = \{x \in E | B(x) \subset X\}$$
$$dilatation_B(X) = \{x \in E | B(x) \cap X\}$$

Partant de deux éléments structurants $B^1(x)$ et $B^2(x)$, [SERR86] définit les HMT, *hit-or-miss transformation*, comme la différence symétrique entre l'érodé de X par $B^1(x)$ et la dilatation de X par $B^2(x)$, c'est à dire:

$HMT(X) = (érodé\ de\ X\ par\ B^1(x)) \cap (dilatation\ de\ X\ par\ B^2(x))^c$.

L'intérêt de cette classe de transformations est d'offrir un appui théorique à une constatation *a priori*, qui suppose que ces transformations sont irréversibles, et diminuent donc la quantité d'information contenue dans chaque image traitée. En fait le terme "analyse" contient de lui-même cet état de fait. Ceci étant, on peut chercher dans l'outil théorique sous-jacent certains résultats qui peuvent aider à comprendre les analyses successives, à les affiner ou les optimiser.

En effet, pour l'instant nous avons considéré un espace image, un espace information et un ensemble d'outils qui permettent de passer de l'image à son contenu. L'analyse morphologique regroupe ces trois objets en un seul: un treillis. Dans ce treillis, peut-être infini, on a un ensemble d'images sur lequel on applique des transformations (de type HMT); et pour chaque couple dual d'une image et d'une transformation appliquée, on obtient une nouvelle image qui contient moins d'information dans le sens de la quantité de pixels, mais plus du point de vue de l'analyse du contenu de l'image. A l'instar des CPO, les transformations successives peuvent avoir un point fixe qui soit l'objet recherché. Ces transformations peuvent s'effectuer en parallèle; certaines disparaissent en cours de route pour privilégier les meilleures, toujours en terme d'information.

Filtres morphologiques

Soit ψ une transformation morphologique, alors elle peut être:

(i) *croissante:* ψ est croissante quand elle conserve l'inclusion, i.e.

$$X \subset Y \Rightarrow \psi(X) \subset \psi(X) \qquad \forall X, Y \in \wp(E)$$

(ii) *anti-extensive:* ψ est anti-extensive quand elle rétrécit X, i.e.

$$\psi(X) \subset X \qquad \forall X \in \wp(E)$$

(iii) *idempotente:* ψ est idempotente quand son application à $\psi(X)$ ne modifie pas ce dernier, i.e.

$$\psi[\psi(X)] = \psi(X) \qquad \forall X \in \wp(E)$$

Soit la classe particulière des transformations HMT où $B^2(x)=\varnothing$, celle-ci représente l'ensemble des érosions [SERR86]. Si $B=B_0$ est l'élément structurant placé à l'origine x=0, alors l'érodé Y de X par B est l'ensemble des points x où le translaté B_x de B_0 par x est inclus dans l'ensemble X. Ainsi [SERR86] remarque que l'érosion ressemble à la soustraction définie sur les ensembles par Minkowski:

$$Y = X \ominus B = \bigcap_{b \in B} X_B \qquad \textit{où } \ominus \textit{ correspond à l'érosion}$$

Si l'on tente de donner un sens "visuel" à cette opération, l'élément structurant se déplace sur notre objet et efface tous les points de celui-ci pour lesquels l'élément centré sur ce point ne se trouve pas complètement dans l'objet. Ainsi cette opération correspond à une érosion, en ce sens qu'elle élimine des points "saillants" pour l'élément structurant choisi. Nous remarquons donc deux caractéristiques essentielles de la démarche: d'abord l'élément structurant est la base d'une dualité pour l'opérateur d'érosion -cet opérateur dépend de celui-ci-, et l'élément structurant possède un centre dont dépend aussi le résultat de l'érosion. Le champs des possibilités est donc vaste, puisqu'il possède deux niveaux de variation: la forme de l'élément structurant et la position de son centre.

Si maintenant on s'intéresse à l'opération complémentaire, qui s'effectue sur le fond de l'image X^c, lorsque qu'on érode celle-ci, alors on s'aperçoit que ce fond vient se remplir des points érodés de X: c'est la dilatation. On a donc:

soit $\tilde{B} = \bigcup_{y \in B} \{-y\}$ *le symétrique de l'élément structurant (par rapport à son centre)*

on a $X^c \oplus B = (X \ominus B)^c$ *, où* $\oplus$ *correspond à l'opérateur dilatation*

et $X \oplus \tilde{B} = \{x \in E \mid B_x \cap X \neq \varnothing\}$

Cette opération est d'autant plus souple que son complémentaire ne l'est pas; en effet, si pour l'érosion l'élément structurant doit se retrouver complètement inclus dans l'objet en un point x, pour la dilatation il suffit que cet élément rencontre un point de l'objet (intersection non vide) quand il est centré en x. Cette ambivalence des deux critères est la source du filtrage morphologique. Ainsi la succession entrelacée de l'élimination de petites particules par érosion et du remplissage de petits trous par

dilatation donne naissance au filtre morphologique.

2.4 L'approche connexionniste

Concernant les modèles connexionnistes, nous conseillons au lecteur curieux l'étude effectuée par Michel Binse [BINS90], et qui propose une revue exhaustive des systèmes connexionnistes depuis les fameux réseaux d'automates cellulaires de Von Neumann jusqu'aux travaux plus récents d'Yves Burnod que nous allons bientôt entr'apercevoir. Pour l'essentiel, nous considérons les systèmes connexionnistes comme révélateurs de la dualité existant entre les modèles de décision et les données pour lesquelles ils sont créés. Dans le modèle à couches, l'apprentissage peut être identifié à la création d'un espace de représentation multi-dimensionnel et la définition d'endomorphismes entre chaque couche: un sous-espace est associé à une couche.

Bien que ce modèle ait des racines linéaires, l'utilisation d'opérateurs semi-linéaires de projection d'un espace vers l'autre, sous-entendu d'une couche vers sa suivante, apporte de meilleurs résultats[1]. L'algorithme d'apprentissage recherche les bonnes valeurs définissant ces projections, pour qu'à une certaine classe d'images d'entrée corresponde un certain vecteur de sortie: il s'agit de trouver les paramètres d'une analyse factorielle en composantes principales. L'analogie pouvait paraître scandaleuse il y a quelques années, elle est dorénavant acceptée.

Cette remarque, qui démontre qu'en fin de compte le neuronal ne déplace que de vieilles connaissances, présente ceci de particulier: le neuronal offre une solution à l'apprentissage. Là où il fallait un statisticien pour décortiquer maints diagrammes et résultats de corrélation, l'espoir est de placer une machine neuronale en face du problème. Hélas, si trouver les bons coefficients est un problème de calcul qui pourrait être résolu par les machines à venir (nous pensons surtout aux projets de circuits numériques de réseaux avec apprentissage intégré [ANN90] [AAIA89]), la difficulté technique se situe au niveau de la recherche du meilleur espace de représentation des données, recherche souvent vaine si ces données n'ont pas fait l'objet d'une étude statistique particulière.

Coincé par son empirisme actuel, le modèle neuronal n'offre que des solutions "bâtardes". En acoustique, on y trouve de bons résultats, essentiellement dus à la bonne linéarité des signaux sonores. En analyse d'images, les seuls résultats semblent encore ne concerner que la reconnaissance de caractères dactylographiés... Nous ne parlons pas des diagnostics médicaux appliqués aux maladies du ventre, dont il est évident, voire notoire, que l'appendicite serait une réponse obtenue dans un quart des cas...

[BURN89] propose le vrai réseau neuronal adaptatif qui existe: le cortex cérébral. Ce cortex est composé de colonnes, chacune possédant une centaine de neurones et appartenant à une aire particulière. Cette décomposition en colonnes d'un problème neuronal permet d'utiliser des "boîtes noires", dont on aura programmé les réactions: actives, passives, inhibitrices, en fonction des entrées [BINS90]. Cette méthode pourrait bien à terme s'imposer dans le microcosme connexionniste, à condition qu'elle n'élude pas une partie du problème de l'apprentissage à l'instar du Perceptron, qui supposait l'existence de telles fonctions pré-définies et qu'il suffisait de combiner sans avoir jamais abordé le moyen de leur définition.

1. Ce paradoxe se retrouve dans le différend qui sépare probabilité et statistique, où la seconde n'est qu'une approximation de la première. La non-linéarité des opérateurs empêche le recours à de bons théorèmes, que ce soit le théorème des projections qui prouve l'existence et l'unicité (l'espace devenant convexe) de la solution, ou le théorème spectral qui met en exergue une représentation de l'opérateur.

Quant aux caractères dactylographiés, [FUKU88] présente un modèle qui nous a semblé digne d'intérêt par l'illustration des difficultés d'une méthode connexionniste pour la reconnaissance d'objets, essentiellement dues aux transformations possibles des objets: homothétie, translation, rotation. Le modèle neuronal n'apprend que la donnée d'une figure; si celle-ci comporte un segment horizontal dans le bas, le réseau n'a pas appris à reconnaître des segments horizontaux, mais plutôt il a appris à reconnaître toute forme vaguement horizontale située en bas de l'image... Dans le même esprit que l'algorithme développé sur la Connection Machine, et que nous avons précédemment décrit, Fukushima présente un algorithme d'apprentissage qui permet la reconnaissance de figures légèrement déformées.

D'une architecture multi-couche classique, l'apprentissage consiste donc à modifier les poids portés sur les liens entre couches connectées. Cependant l'adjonction de cellules, qui détectent une même primitive mais décalée par rapport à celle que repère la cellule qui leur est connectée ensuite, insensibilise le réseau à cette forme particulière de bruit que créent ces isométries. Fukushima commente ce mécanisme dans ces termes:

> *"Tolerating positionnal error a little at a time at each stage, rather than all in one step, plays an important role in endowing the network with an ability to recognize even distorted patterns."*

Une autre issue au connexionnisme nous semble profitable, il s'agit du lien entre approches symbolique et connexionniste tel que le décrit [NATO88]. Dans cet article relativement général, l'auteur tente de rapprocher l'analyse structurelle et les méthodes connexionnistes. Les descripteurs retenus pour l'image sont des descripteurs de courbure: pointe-concave, très-concave, concave, droit, convexe,..., associés à un descripteur de longueur: quasi-nul, très-court, court, moyen,... Après vectorisation de la figure, les descripteurs sont analysés globalement par un réseau connexionniste dont le rôle est la séparation des données apprises, puis à reconnaître, en plusieurs classes.

Le connexionnisme est utilisé comme moyen associatif de traitement de l'information symbolique, et non pas de l'information pixel. La nuance est faible, mais la démarche apparaît plus prometteuse comme l'indique l'engouement de l'approche cellulaire pour les problèmes de recherche de l'information (cf. l'utilisation de la Connection Machine pour la recherche de documents dans le domaine financier).

3 L'approche cellulaire

Nous présentons dans ce paragraphe une approche cellulaire qui tente de synthétiser la transformation de Hough et le filtrage morphologique, pour définir une méthode approximative mais rapide d'acquisition de lignes orientées. Cette approche cellulaire rejoint des observations relatives au système visuel chez l'homme, et surtout les travaux de D.Walters concernant la sélection de primitives visuelles à partir d'expériences psychophysiologiques.

3.1 La vision chez l'homme

3.1.1 Digressions

Le langage est pour les philosophes une des sources incontournables de la conscience chez l'homme. Puisque ce langage est la source de toute communication, dont notre époque comprend mieux que toute autre combien elle favorise les connaissances, il semble raisonnable de s'interroger sur les liens qui existent réellement entre les moyens de communication, et leurs possibilités, et le phénomène de conscience qui caractérise, paraît-il, l'homme. De fait, la vision et l'ouïe sont ces outils merveilleux qui permettent à l'homme de communiquer avec le monde extérieur. Et, si ces outils se développent certes plus rapidement chez l'homme que l'outil langage, d'aucun sait d'expérience qu'ils continuent néanmoins d'évoluer[1].

Le côté particulièrement attachant de la vision, consiste en ce rapport étroit qui lie les mots et les choses qu'ils désignent. Face aux abstractions nécessaires, la vision permet d'exciter *in petto* certaines parties de notre mémoire de manière plus directe et plus sensible que les mots eux-mêmes. On ne peut alors s'empêcher de citer la "petite madeleine", qui, bien que mise sur le même plan impressionniste que les "pavés de la Place Saint-Marc", offrit à Proust ses plus belles pages mais aussi son plus fidèle souvenir d'un Temps lointain, recherché mais révolu. Parce que cette recherche en valait bien d'autres, ce dilettante génial s'enquit d'une poursuite inexorable du Temps. Et le lecteur retient de cette longue quête originale la conscience de ces liens, qui entre mots et images, entre sons et sensations, décrivent et limitent notre pensée, nos actions et notre perception.

A la façon des deux côtés qui hantent la Recherche, tout homme construit sa perception du monde entre les sensations qu'il y puise, et les noms qu'il leur donne. Les liens créés entre certains noms dépendent de la corrélation des sensations perçues lors des rencontres avec ces objets, cette matière brute. Marr l'a d'ailleurs rappelé:

> *"I begin to sea more clearly the force of the idea that perception is the construction of a description."*

Le processus de vision ne doit plus alors être considéré par la succession de ses traitements, ce qui, toujours à la façon de [MARR82], correspondrait à une analyse au niveau du transistor d'une machine qui effectue une transformée de Fourier; analyse impossible compte tenu de la complexité du traitement, et du manque total d'intérêt d'une telle description. Au contraire, la vision doit être perçue dans un cadre complet, à la recherche de cette compréhension inéluctable du processus naturel qui nous offre *notre* vue de ce monde, identique dans ses aptitudes et tellement variée dans la construction symbolique qu'elle

1. S'il en fallait une preuve, je ne citerais que cette anecdote délicieuse d'un jour où je ne sus contenir ma surprise d'avoir entendu ma nièce nommer *chien*, ce que je savais être une *vache*! Cette surprise contenait alors toute l'expression naïve d'une pensée trop entière, à cause de son jeune âge, et incapable d'identifier le mécanisme qui avait pu rapprocher deux animaux qui me semblait alors trop éloignés visuellement.

crée à un niveau supérieur, celui justement de la symbiose des sensations et des noms, celui des images eidétiques de Proust.

Ceci dit, nous ajoutons que les traitements que nous venons de décrire se dirigent vers un même objectif, la description sous formes de segments d'une figure, et que notre volonté n'est pas de rejeter cet objectif mais au contraire de lui apporter une solution matérielle adaptée. L'oeil ne se suffit pas d'un traitement par scène, il en fait un toutes les centaines de milli-secondes environ. Pourquoi ne pas chercher à l'imiter, et proposer une solution qui ne fonctionne certes pas en 100 ms car ceci n'est qu'une question technique, mais offre le résultat d'un traitement en temps *quasi*-linéaire, quitte à perdre quelques mesures quant à la qualité du traitement?

3.1.2 Le système visuel

C'est à partir d'observations psychophysiologiques, que [WALT87] propose de chercher un ensemble de primitives visuelles qui puissent servir de base à la construction d'une machine de vision générale. Car, les essais actuels de vision générale butent encore sur la définition de telles primitives. La *vision du monde* [MARR82], restreinte dans le cas d'un robot industriel manipulant des objets simples [POST87], s'élargit infiniment pour l'oeil humain. Toute la difficulté se situe donc à cette frontière inévitable et nécessaire qu'est la définition des primitives visuelles que manipulera le système de vision.

> *"There exist many theories of shape description and recognition, each attempting to explain some specific aspect of the problem. This is so because it is possible to conceptualize shape as a high-level perceptual function. Since there is very little neurophysiological evidence about its nature and we are not really sure what the basic constituants are, the field has been open to freewheeling hypothesization"*, extrait de [LEVI85].

Deux tâches se partagent de façon duale le domaine de la vision générale: l'extraction de contours significatifs, et la détection de régions. Extrait de [WALT87], l'exemple de la figure 7.a démontre que l'oeil n'évalue pas les droites comme aussi significatives en terme d'information qu'une courbe fermée. Bien que celles-ci aient toutes la même intensité, la courbe nous[1] semble ressortir de cette figure.

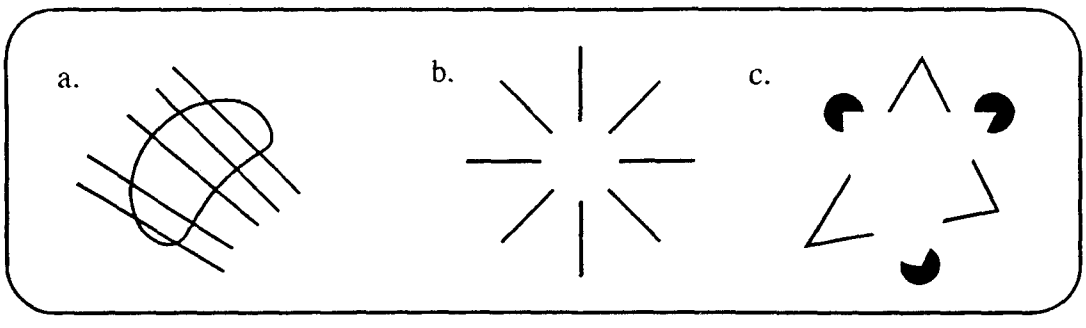


figure 7 illusions visuelles qui illustrent certaines propriétés de la vision générale

Partant de là, la figure 7.b met en exergue un phénomène corrélatif. Puisque l'oeil privilégie les contours, cet exemple montre combien il est facile de le tromper: notre oeil nous suggère un contour inexistant. Ce phénomène est connu sous le nom d'illusion d'Ehrenstein; la figure 7.c donne le classique du genre que le lecteur retrouvera sur la couverture de [MARR82].

1. Le *nous* n'est pas explétif: nous lui donnons le sens d'une interpellation du lecteur qui confirmera ou infirmera ainsi nos dires, puisque chacun peut en juger facilement et que nous ne sommes doué d'aucune empathie particulière.

[RESN89] analyse ces constructions subjectives de contours, qui indiquent que: "'*the creative filling in' of omitted information in images is routenely performed as image information is processed*'. Les études menées sur des singes accréditent cette idée de simultanéité des processus de reconnaissance des formes existantes, et de génération spontanée de contours inexistants [HEYD84]. D'ailleurs cette création, puisque tel est le cas, interviendrait à un niveau relativement bas du processus de vision, dans l'aire 18 du cortex visuel du macaque. Cette aire visuelle, aussi référencée comme V2 [BURN88, page148], Thorpe en explique le fonctionnement [THOR88].

Partant des 125 millions de photorécepteurs qui composent la rétine [RESN89], l'information visuelle subit deux étapes de traitements dans cette même rétine, puis est envoyée vers le cerveau *via* le nerf optique. Après passage dans le noyau géniculé, l'information parvient au cortex visuel, encore appelé aire 17 ou V1. De là, les messages se propageront vers les aires suivantes: V2, V3, V4, MT... De là, ceux-ci sont dirigés soit vers le cortex inférotemporal, impliqué dans l'identification des objets, soit vers le cortex pariétal, qui permettrait l'analyse des configurations spatiales des objets [THOR88].

Ainsi les différentes composantes d'une image sont traitées chacune par des aires spécialisées, que ce soit pour les couleurs, l'orientation, ou le mouvement. Une fois traitées par ces neurones particulièrement sensibles, les composantes analysées sont regroupées pour l'identification des objets et la recherche de la configuration spatiale.

Quant à l'aspect temporel de ces traitements, [THOR88] remarque que le traitement visuel prend entre 100 et 150 milli-secondes, parmi lesquelles les trente à quarante premières ne concernent pas le cortex visuel. Ceci signifie qu'entre l'arrivée des messages dans l'aire V1 et leur sortie du cortex visuel (après l'aire d'identification des objets), les neurones des différentes couches n'ont que soixante à soixante-dix milli-secondes pour analyser l'image. [THOR88] en conclut l'impossibilité, ou tout du moins la forte improbabilité, de traitements du type *feedback*. Cette question se pose à tous, puisqu'il y a autant de fibres descendant du cortex visuel vers le noyau géniculé qu'en sens inverse. De plus, maints traitements font appel aux méthodes itératives en vision; elles sont d'ailleurs souvent lourdes, comme la relaxation qui demande plusieurs centaines d'itérations de boucles.

Pour clore là cette discussion, [THOR88] joue Cassandre au pays des candides:

> *"Dans un certain sens, ceci doit donner espoir aux chercheurs en traitement d'images: si le cerveau peut identifier des objets avec 10 ou 15 étapes de traitement, sans utiliser ni codage analogique précis, ni boucles itératives, il devrait être possible d'arriver au même résultat avec des systèmes artificiels."*

Pour revenir au point abordé précédemment sur la subjectivité de l'oeil dans le processus de vision, [WALT87] propose de mesurer les contrastes dans l'aperçu de formes simples. Ainsi dans la figure 7.b & .c, les régions inexistantes surgissent sur le fond par un effet de contraste qui n'est que virtuel. De même, à la figure 8, le lecteur distinguera par lui-même combien l'oeil semble donner plus de contraste à l'objet *a* qu'à l'objet *b*.

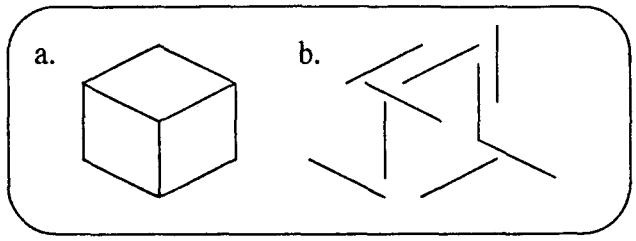


figure 8 évaluation des contrastes

De ce constat fragile, mais corroboré par des études fines sur cette perception des contrastes chez l'homme, [WALT87] tire quelques conclusions dont nous en retenons deux essentielles. Elles concernent la longueur des lignes, et les relations entre les terminaisons des lignes. Pour la longueur des lignes, une expérience psychophysiologique démontre qu'une ligne de longueur 60 minutes est une bonne base pour détecter des différences de contraste avec d'autres lignes, qui sont pourtant dessinées dans la même direction et de même largeur. Les lignes trop longues se discernent plus difficilement. Ainsi se dégage l'idée d'un opérateur locale de détection de lignes relativement petites, environ 1 degré d'arc visuel.

Pour ce qui est des relations entre les terminaisons des lignes, [WALT87] rapporte une expérience avec 5 observateurs qui évalue l'influence des connexions entre lignes, c'est à dire de leur relation, sur la perception de celles-ci. Comparant divers types de formes, on obtient le diagramme suivant:

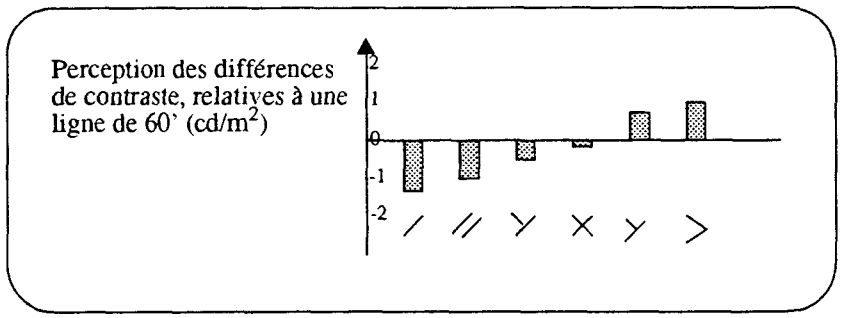


figure 9 résultat d'un test extrait de [WALT87], démontrant l'influence des formes connectées

Les valeurs positives indiquent que la forme est apparue plus contrastée que la ligne de référence choisie; les valeurs négatives l'inverse. De ce diagramme, nous déduisons que les lignes connectées entre elles apparaissent plus contrastées que celles sans relation. Cette déduction est relativement importante, intuitive pour certains, et cohérente[1] avec ce que nous avons présenté de la "machine visuelle" de l'homme.

1. Le manque de contraste des lignes parallèles de la figure 9, s'expliquerait, selon [WALT87], par cette hiérarchisation des processus qui verrait le parallélisme traité à un niveau global. De cette façon, le contraste n'étant qu'une caractéristique locale de l'image, ces lignes ne sont pas mises en valeur par l'oeil. Un processus de plus haut niveau se charge simplement de détecter leur parallélisme. Au contraire, les intersections de lignes, et surtout leur connexion, se renforcent considérablement puisqu'elles contiennent certainement une information essentielle.

3.1.3 Le ρ-espace de représentation

Laissons à d'autres le soin de conclure cette analyse succincte de notre système visuel, et d'entrevoir le déroulement de l'exposé:

> *"The human visual system is designed to produce organized perception. Information consisting of a variety of such spatial features as size, shape, distance, relative position and texture is structured by the mind to represent visual scenes. These spatial features are perceived as properties of things, objects in the scene, and not merely as abstract lines or surfaces...*
>
> *We do not perceive lines or unattached extents; we perceive objects"*, extrait de [LEVI85].

L'acquisition des primitives d'une image ne peut donc se faire sans une approche globale cohérente. Que l'on extraie des éléments significatifs, tels que des segments, des arcs de cercle,..., ne doit pas se faire sans que cette acquisition d'information soit globale: le segment est-il connecté?, l'arc de cercle est-il seul?,... Dans cette perspective, les informations locales conservent un caractère global qu'il faut mettre en exergue.

D'ailleurs, si l'on se tourne vers un mécanisme de vision bas-niveau, découvert par Hubel et Wiesel, et qui leur valut un Nobel, le cortex visuel contient des neurones sensibles aux segments orientés, avec une définition d'environ 10^{o} pour chaque orientation. Les segments orientés sont du même type que ceux donnés à la figure 10. Pour plus de détails, le lecteur se reportera à [LEVI85] aux chapitres 5 & 8, notamment la figure 5.4 page 158, ou à [HUBE79].

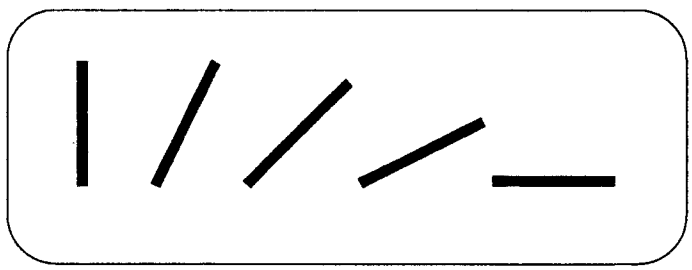


figure 10 segments orientés, auxquels sont sensibles certains neurones

De ces connaissances, [WALT87] dérive un espace de représentation sur lequel chaque image verra son contenu projeté. Cet espace est intermédiaire entre l'image même et la description sous forme de lignes de celle-ci. Le ρ-espace possède trois dimensions: deux pour matérialiser le plan, une pour représenter les orientations. Il faut souligner que dans le cortex visuel, si certains neurones sont sensibles à une orientation, ils ont en plus une position définie que caractérise l'arc visuel traité par chacun des neurones. Aussi la figure 11, qui représente ce ρ-espace, projette de manière abstraite nos soupçons quant au système visuel humain (vaste oeuvre d'anthropomorphisme...).

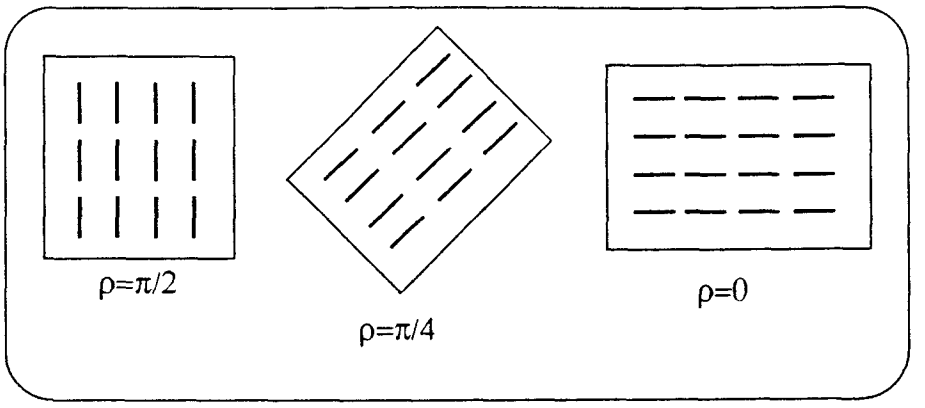


figure 11 ρ-espace pour ρ=3

Chaque image y est projetée, et donc décomposée en petits segments dans un espace 3D. Sur chacun des plans de cet espace, on trouve des segments orientés dans une direction donnée et répartis sur tout le plan. Chaque plan considéré correspond à l'image tournée d'un certain angle. La troisième dimension de l'espace de représentation est parcourue par les différents angles de rotation de l'image de départ. Une recomposition d'image, par exemple l'approximation polygonale d'un carré dans une image, s'obtiendra par parcours successifs des différents plans et recolement des petits segments colinéaires de chaque plan.

3.1.4 Vers une approche cellulaire

L'architecture la plus adaptée à ce type de représentation semble bien être l'architecture cellulaire. A la manière du Perceptron, les différentes zones de l'image sont connectées à un ensemble de cellules, à schéma de communication et de décision local. Chaque groupe de cellules est un opérateur, qui recherche les segments orientés. Cet opérateur fonctionne comme une convolution avec des noyaux distincts; il existe autant de noyaux que d'orientations cherchées. Cette architecture est proposée par [WALT87]; le nombre d'orientations est de 16 et les noyaux ont une taille de 7 sur 7 pixels.

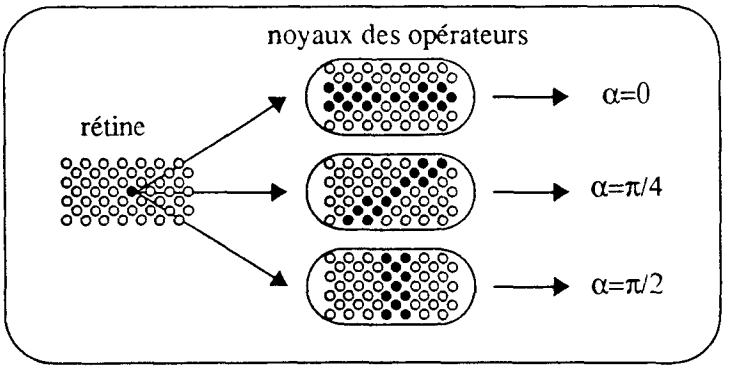


figure 12 Opérateur proposé par [WALT87, p.275] pour l'acquisition des segments orientés

La détection de caractéristiques orientées se retrouve assez souvent dans la littérature concernant les méthodes cellulaires. Nous avions cité au chapitre 1 un algorithme de tracé de segments qui utilise le caractère discret du nombre d'orientations de l'espace image pour un fonctionnement parallèle [STAM75].

3.2 L'approche segments orientés

3.2.1 Historique de nos errements

Il convient ici de rappeler qu'au début du projet PASTIS, nous avions pour point de mire une étude des réseaux connexionnistes appliqués à la reconnaissance des formes. A dire vrai, l'intérêt que nous portions alors au connexionnisme tenait plus dans les idées fortes qu'il semblait embrasser: parallélisme, associativité, calcul approché,... Toutefois, avec l'esprit trop cartésien qui nous caractérise, le pragmatisme régnait et nous avons longtemps cherché un modèle neuronal adapté à nos besoins. Sans succès.

Pour obvier à toute remarque désobligeante quant à notre volonté de réussir, nous avons tenté une approche systématique, mais naïve, de l'analyse bas niveau d'une image. Michel Binse proposa un alphabet qui rendrait compte des différentes composantes de l'image. Ces composantes sont celles que nous avons retrouvées plus tard chez [WALT87]. Elles indiquent toutes les connexions et terminaisons de petits segments dans une image. Le langage ICLYXi naissait! Voici sa représentation:

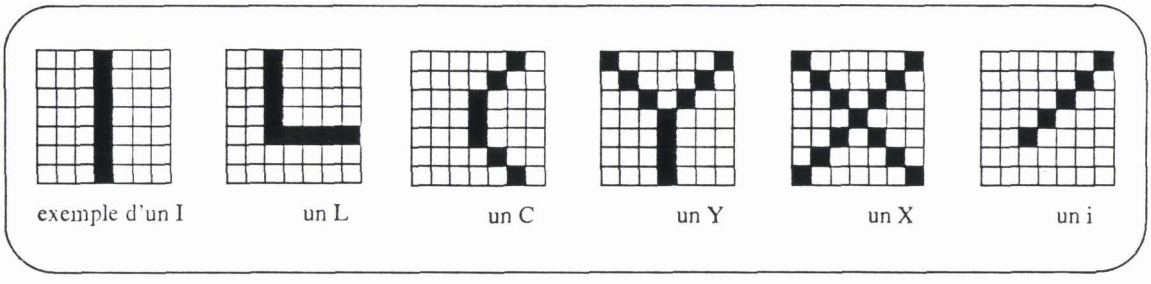


figure 13 le vocabulaire ICLYXi

Se doter d'un tel alphabet peut paraître saugrenu. Il procède néanmoins d'une idée générale qui nous a séparé de l'apprentissage "réseau de neurones". Dans l'apprentissage associatif, on tente de reconnaître d'emblée l'image: le réseau se charge de la décomposition, en interne et de manière transparente... si ça marche. La mise en oeuvre d'un alphabet détruit une certaine forme d'associativité qui devrait exister entre les mots, le vocabulaire, et leur sens, la sémantique. De plus naïfs, nous revenions à l'existant, c'est à dire aux résultats de l'analyse structurelle et à ceux de [WALT87][1].

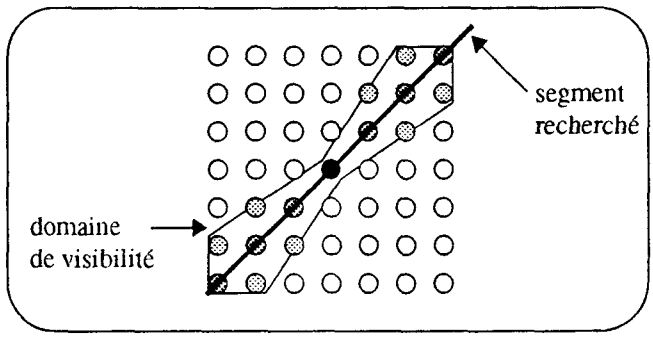
L'approche cellulaire nous a d'abord tenté, mais nous voulions lui donner une direction numérique. En effet il ne nous a jamais semblé satisfaisant de ne mettre en oeuvre que des méthodes cellulaires, de type convolution, ou morphologiques. D'abord ces méthodes doivent être repensées pour une projection matérielle. Le cellulaire connaît ces limites dans les machines pyramidales, qui souffrent le plus souvent d'un goulet d'étranglement dans les échanges vers le haut. Ensuite, si l'on se rapproche par dualité de ce qui se fait en synthèse d'images, force est de constater que la mesure métrique des objets reste incontournable; que ce soit pour identifier une primitive plus élaborée, ou pour positionner les objets entre eux:

> *"All parts of each object are perceived together in one construction - not as separate, independant and freefloating elements. And all objects are perceived as related to each other near, far, behind,..."* tiré de [LEVI85], citant [HABE82].

1. Cette remarque me tient à coeur, car d'une part nous n'avions pas alors connaissance de ces travaux, mais bien plus important, cette "redécouverte" correspondait à un leitmotiv favori du groupe, dont l'objectif était de réaliser un amalgame d'essais-erreurs "juste pour voir"...

Notre démarche originale consiste à avoir cherché les approximations possibles, réalisables, qui permettent de réduire le coût de traitements, considérés comme indispensables en vision. Ainsi, lors même que les segments orientés formeraient ou non une bonne classe de primitives pour l'analyse d'image, nous avons essayé de les extraire rapidement, et avec toute la souplesse qui caractérise une machine d'évaluation. Car le second défaut des architectures cellulaires se situe dans leur aspect figé: une cellule de [WALT87], une fois câblée, détecterait un nombre défini de directions, sans possibilité d'en ajouter. L'une des possibilités qui s'offrent alors à l'utilisateur, serait de tourner l'image à traiter d'un angle inférieur à la résolution cellulaire.

Partant de cette idée, deux remarques s'imposent. D'une part la rotation reste une opération relativement coûteuse en temps; nous le verrons au prochain chapitre. D'autre part, la résolution du réseau cellulaire est finie. Elle est notamment limitée par l'étalement, sur un voisinage, du segment à détecter.



Ce domaine de visibilité du segment orienté limite nécessairement la résolution du système cellulaire, une fois celui-ci câblé. Le cellulaire apparaît donc une bonne solution lorsque toutes les caractéristiques de la détection sont connues: primitives, taille de celles-ci, résolution,... Mais, comme nous l'avons vu pour la Transformée de Hough, la variabilité des paramètres est souvent un critère essentiel pour l'amélioration des performances des opérateurs. Dans le cas de la TH, le critère porte sur une meilleure visibilité des pics en fin de traitement. Dans le modèle vers lequel nous tendons, et dont le lecteur pourrait déjà pressentir combien il sera proche de cette transformation, la modulation de certains paramètres procurera un avantage incontestable à cette méthode.

Toutefois, en faisant l'exégèse de ces dernières lignes, si nous avions la possibilité de mettre en oeuvre une rotation rapide, la détection des segments orientées se ferait facilement: en tournant successivement l'image de l'angle voulu (résolution laissée à la diligence de l'utilisateur), puis en extrayant uniquement les segments horizontaux sur chacune des images tournées. Le double avantage de cette méthode consiste en la résolution non fixée *a priori* du mécanisme de détection de segments horizontaux: celui-ci est facilement réalisable puisqu'il s'apparente à un produit de convolution. De plus, pour être efficace, il n'a pas besoin d'une réalisation cellulaire. Un tel mécanisme de détection pourrait être implanté à l'aide d'une architecture micro-programmée, offrant toute possibilité de résolution. La difficulté se situe donc au niveau de l'opérateur de rotation.

3.2.2 La rotation rapide

Nous présentons au chapitre 3 les divers travaux et réalisations concernant la rotation. Il s'agit pour l'essentiel de la réalisation de circuits VLSI rapides, ou du développement de la rotation sur machines parallèles (SIMD ou MIMD). Pour la rotation, le calcul des coordonnées est toujours effectué sur des nombres réels, de manière précise, et les problèmes soulevés proviennent le plus souvent de la fonction d'interpolation, qui permet de passer de ces nombres réels aux nombres entiers, indices de la position des pixels dans le plan image.

Le temps requis pour tourner une image est alors dû en grande partie à ce calcul réel, et donc précis, des nouvelles coordonnées, mais aussi à la quasi-impossibilité de trouver un parallélisme quelconque à ce calcul sans engendrer des goulets d'étranglement sur la mémoire d'image, surtout en sortie. Quant à la solution que préconise [KUNG82] pour les problèmes limités par les entrées-sorties:

> *"It should be clear that any attempt to speed up an I/O-bound computation must rely on an increase in memory bandwidth. Memory bandwidth can be increased by the use of either fast components (which could be expensive) or interleaved memories (which could create complicated memory management problems)."*

La première nous l'écartons de fait; la seconde, qui consiste à construire une mémoire d'images à bancs mémoire entrelacés, elle peut s'avérer inefficace car une suite de pixels, dont les adresses seraient contiguës au départ, ne garantit nullement après rotation une combinaison fixe quelconque entre les nouvelles adresses.

Pour reprendre [KUNG82], il faut noter que selon sa définition, la rotation n'est pas un problème limité par le nombre d'entrées-sorties, *I/O-bound computation*, mais plutôt limité par les calculs, *compute-bound computation*. En effet, pour chaque pixel lu, la rotation de celui-ci demande quatre multiplications et deux additions. Ceci indique, toujours selon [KUNG82], que la rotation doit pouvoir s'exprimer sous une forme systolique... qu'il reste à définir. Nous pensons que la proposition architecturale de [MOUG86][1] va dans ce sens.

Pour notre part, il nous a semblé intéressant d'orienter notre réflexion sur la "parallélisation" même de l'algorithme de rotation, en considérant que nous avions virtuellement un réseau de calcul suffisamment puissant pour supporter les six opérations arithmétiques effectuées pour chaque pixel. Sous cette hypothèse, la limite algorithmique se déplace vers le problème du nombre d'entrées-sorties. Si l'accès mémoire en entrée peut se faire par blocs de pixels consécutifs, en sortie les pixels tournés auront perdu cette organisation consécutive en faveur d'une disposition aux quatre premiers voisins. Il est dès lors difficile de trouver un parallélisme quelconque pour le rangement en mémoire de sortie des pixels, sinon à modifier l'algorithme.

3.2.3 Décomposition d'une rotation

Au deuxième chapitre, nous avons vu que les machines SIMD à communication réduite aux quatre premiers voisins satisfaisaient parfaitement les critères matériels de complexité d'intégration, de simplicité du contrôle et d'échange de données. Ces échanges peuvent s'effectuer sur un nombre important d'éléments, un par PE, et sur un cycle machine. Or, l'organisation aux quatre premiers voisins correspond à celle d'une ligne horizontale quelconque obtenue après rotation d'un angle inférieur à $\pi/2$. Elle est calculée selon l'algorithme de Bresenham. Nous avons donc essayé de mettre en pratique ceci, en remarquant de plus qu'une ligne tournée se déplacerait alors facilement dans un réseau maillé à quatre voisins, *mesh network*.

Pour ce faire, il fallait trouver un lien entre la possibilité de déplacement et l'algorithme de rotation lui-même. Ce lien, nous l'avons retrouvé simplement dans la décomposition géométrique de la rotation d'un point (x,y) par rapport à l'origine (x_0,y_0). En effet, on montre facilement que cette rotation se décompose en une rotation de (x,y) par rapport au centre (x_c,y_c) de la ligne parallèle à l'axe Ox qui le

1. Que le lecteur se rassure, nous en reparlerons plus amplement.

porte, et une translation du point obtenu par un vecteur donné par la rotation de (x_c,y_c) par rapport à (x_0,y_0).

Voyons cela:

la notation matricielle de la rotation est: $\vec{P}' = R \cdot \vec{P}$

avec $R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$ $\qquad X' - X_0 = R \cdot (X - X_0)$ (3.3)

et $\vec{P} = \begin{bmatrix} x - x_0 \\ y - y_0 \end{bmatrix} = X - X_0 \qquad \vec{P}' = \begin{bmatrix} x' - x_0 \\ y' - y_0 \end{bmatrix} = X' - X_0$

c'est à dire:
$$\begin{cases} (x' - x_0) = \cos\theta \times (x - x_0) - \sin\theta \times (y - y_0) \\ (y' - y_0) = \sin\theta \times (x - x_0) + \cos\theta \times (y - y_0) \end{cases}$$

Soit (x'',y'') le point obtenu par rotation de (x,y) autour du centre (x_c,y_c) de la ligne horizontale qui le porte, il vient:

$$\vec{P}''_{(c)} = \begin{bmatrix} x'' - x_c \\ y'' - y_c \end{bmatrix} = R \cdot \vec{P}_{(c)} = R \cdot \begin{bmatrix} x - x_c \\ y - y_c \end{bmatrix} \qquad \text{avec } y = y_c$$

ce qui peut s'écrire $X'' - X_c = R \cdot (X - X_c)$

qui, pour toutes les lignes horizontales constituant l'image ($x_c = 0$, $0 \le y_c < N$), donne le même vecteur résultat pour tous les points (x,y) de ces lignes situés à la même distance $\delta = x - x_c$ du centre de leur ligne. Nous avons donc là une première forme de parallélisme, qui limite pour les N^2 pixels d'une image le calcul de cette rotation à seulement 2N valeurs différentes[1] (et non $2N^2$).

Nous avons:

$$X'' - X_c = R \cdot (X - X_c)$$

donc, $\quad X'' - X_c = R \cdot (X - X_0) + R \cdot (X_0 - X_c)$

et $\quad R \cdot (X - X_0) = X'' - X_c + R \cdot (X_c - X_0)$ (3.4)

d'où, des équations (3.3) et (3.4) nous déduisons:

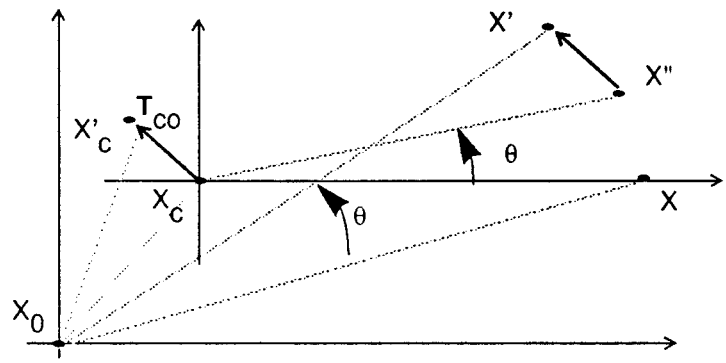
$$X' - X_0 = X'' - X_c + T_{co}$$

où
$$\begin{cases} X'' - X_c = R \cdot (X - X_c) & \textit{rotation de } (x,y) \textit{ par rapport à } Ox_c \\ T_{co} = R \cdot (X_c - X_0) & \textit{translation fixée pour une ligne} \end{cases}$$

1. la donnée 2N correspond à N valeurs $x'-x_c$, et N valeurs $y'-y_c$.

Une rotation se décompose donc en une rotation de centre celui de la ligne tournée, suivie d'une translation d'un même vecteur T_{co} pour chacun des pixels tournés de cette ligne. Le fait que cette translation s'applique à l'ensemble des pixels d'une même ligne correspond bien à ce que nous voulions obtenir comme déplacement global limité aux quatre premiers voisins.

La figure géométrique ci-dessous illustre les équations précédentes:



Notons que le vecteur translation T_{co} correspond au vecteur joignant les points (x_c,y_c) et (x'_c,y'_c), respectivement le centre de la ligne horizontale à laquelle (x,y) appartient, et le point obtenu par rotation de ce centre par rapport à l'origine de l'image (x_0,y_0).

Nous présentons au chapitre 3, une réalisation d'un processeur rapide de rotation utilisant une approche systolique qui s'applique bien à ce type de parallélisme (*mi*-parallélisme, *mi*-pipeline). De plus, le gain obtenu par cette parallélisation du processus de rotation est contrebalancé par la perte de précision de la rotation telle que nous l'avons réalisée de manières logicielle et matérielle. En effet, et bien que nous le détaillions par la suite, nous pouvons d'ores et déjà remarquer qu'une réalisation en deux temps d'un tel processus implique la combinaison malheureuse de deux interpolations successives. Ce doublet de l'interpolation induit nécessairement une perte de précision, et donc le risque de perdre la connexité des pixels.

3.2.4 Détection de segments orientés

Deux méthodes possibles

Une fois obtenue la parallélisation du processus de rotation, la description en segments orientés s'obtient par succession du processus "rotation-extraction segments horizontaux", où la rotation est réitérée sur un angle faible (de l'ordre de $\pi/16$ ou $\pi/8$) jusqu'à π. L'extraction de segments horizontaux peut se faire de différentes manières, celles précisées dans le *paragraphe 2*. [LAHA86] distingue globalement deux méthodes:

(i) le suivi de contour, et la segmentation,

(ii) la convolution de l'image par un modèle.

Le suivi crée un chaînage des pixels susceptibles d'appartenir à un même contour. Sur chaque chaîne obtenue est appliqué ensuite un algorithme d'approximation de courbes, le plus souvent approximation polygonale. Comme le remarque [LAHA86], cette méthode est très sensible à la qualité des images, et tout particulièrement à la présence de *trous* dans ces lignes de contour. Autre inconvénient, cette technique est difficilement parallélisable: [GALL86] aborde ce problème. L'algorithme de suivi voit sa complexité croître avec la complexité de l'image.

A l'opposé, plutôt que de rechercher une description de l'image, les techniques de convolution prennent comme modèle la forme recherchée et tentent de l'apparier dans toute l'image. L'algorithme cherche alors la zone où cet appariement devient maximum, signifiant la présence possible de la forme recherchée dans cette zone. Bien qu'elle nécessite de nombreux calculs, cette méthode apparaît plus efficace puisqu'elle peut être calculée de manière parallèle par un processeur câblé. De plus, une convolution est relativement insensible au bruit, et aux trous qu'aurait forcément générés notre rotation rapide. La transformée de Hough se classe ici dans les techniques de convolution, puisqu'elle travaille sur un modèle de formes.

Toutefois ces techniques de convolution ne donnent pas une classification structurelle pure des objets contenus dans l'image. Elles précisent l'existence probable d'un objet, dont soit la convolution renforce -en terme de pixels- la présence, soit la TH propose les paramètres; mais elles n'en donnent pas une description structurelle. En effet, avec les informations obtenues, la recherche d'une structure particulière passera par la mise en oeuvre d'une segmentation, qui se révelera alors beaucoup plus rapide.

Dans le cas qui nous intéresse, nous allons d'abord appliquer une convolution pour ne plus travailler sur des pixels, mais sur des blocs de pixels ayant une forme proche du petit segment: ici un bloc rectangulaire de hxl pixels, où la largeur l sera beaucoup plus importante que la hauteur h. Ensuite, sur les blocs qui se retrouveront "allumés" par un processus que nous décrirons, nous appliquerons une sorte de TH horizontale, en évaluant pour chaque ligne de blocs horizontaux le nombre des leurs qui sont allumés. Quand ce nombre se révèlera supérieur à un certain seuil, nous irons les chaîner entre eux, puis les regrouper à la manière d'une approximation polygonale d'une ligne horizontale. De cette ligne, nous aurons toutes les indications nécessaires: extrêmités, longueur, angle de rotation dans l'image.

On le voit cette méthode mèle à la fois convolution, TH et approximation polygonale (bien que pour cette dernière nous n'ayons pas recours au critère des moindres carrés). Nous allons décrire la convolution que nous avons choisie pour extraire les petits segments orientés.

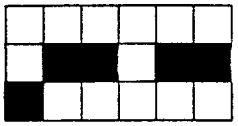
Quelle convolution ?

La convolution est une technique coûteuse en calculs. Pour la réduire nous avons voulu à la fois diminuer le nombre de points sur lesquels nous l'effectuions, et reconsidérer le mode de calcul. En effet la convolution est évaluée pour chacun des points de l'image, ce qui donne dans le cas du dessin au trait une résolution trop fine puisque le nombre de pixels noirs est souvent de l'ordre de 1% du nombre total de pixels. Cette approche provient de ce que dans la convolution, l'élément de base est le pixel: on prend un pixel, on regarde son voisinage et on modifie le pixel en fonction. Dans le dessin au trait, cette méthode ne repose sur aucune réalité car les dessins sont généralement grossiers; les traits sensés représenter des segments ont "globalement" l'apparence d'un segment. Aussi, une fois évalué l'état d'un voisinage, nous avons voulu conserver cette notion de voisinage, cf. notre bloc de pixels, qui nous a semblé plus proche de la réalité du dessin étudié.

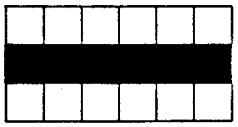
De plus, dès lors que cette convolution n'est plus appliquée à l'ensemble des pixels mais à une partition en voisinages de cet ensemble, l'algorithme de convolution tel que nous l'avons décrit au *paragraphe 2.3.2* n'est pas forcément le plus adapté. De même, la proposition cellulaire de D.Walters, dont nous avons noté la similitude avec une convolution au *paragraphe 3.1.4*, apparaît inadéquate pour notre réalisation. La technique qui s'est rapprochée finalement le plus de nos besoins, est le filtrage morphologique, notamment avec la dilatation et l'érosion.

Hélas, bien que l'aspect ensembliste réponde en partie à nos vues, la dilatation possède un critère faible[1]: que l'intersection soit non vide, et l'érosion un critère trop difficile: en ce sens qu'il y a inclusion du schéma du bloc de pixels considéré dans l'élément structurant. A partir de la configuration repré-

sentée par le bloc suivant:



nous voudrions obtenir le schéma suivant:



gardant à l'esprit que le dessin peut présenter des trous ou des imperfections qu'il faut traiter au niveau bas de l'analyse. On remarque aussi que la notion de centre de la transformation n'a ici que peu d'importance: on passe d'une certaine disposition d'un voisinage à un schéma donné (ou à un schéma vide si la disposition initiale n'a pas vérifié le critère de transformation).

Nous avons donc cherché à quantifier ce critère de ressemblance à un schéma donné, qui serait ici notre élément structurant. Or, pour quantifier les relations topologiques, la notion de voisinage est une bonne approche. Pour cela, appelons I l'espace image, c'est à dire l'écran, et $\wp(I)$ l'ensemble des parties de I, donc toutes les figures possibles sur cet écran. L'application $\nu_{k,r}$, donnée de la façon suivante:

$$\nu_{k,r} : \wp(I) \rightarrow \wp(I)$$
$$X \rightarrow \nu_{k,r}(X)$$

$$où \quad \nu_{k,r}(X) = \{ x \in X \,/\, la\ boule\ B(x,r)\ possède\ au\ moins\ k\ points\ de\ X \}$$

définit un voisinage sur $\wp(I)$. On remarque au passage que $\nu_{k,r}$ n'est pas extensive: la figure X n'est pas complètement incluse dans son voisinage $\nu_{k,r}(X)$ puisque les points isolés de X n'y sont pas. Par contre cette applications est croissante, dans le sens que nous lui avions donné au *paragraphe 2.3.3*, c'est à dire que si une figure X est incluse dans une autre Y alors le voisinage de X: $\nu_{k,r}(X)$, est inclus dans celui de Y: $\nu_{k,r}(Y)$.

Voisinage approchée

Ce voisinage va nous servir à bâtir le nôtre. En effet, plutôt que de partir d'un élément structurant qui aurait une certaine forme, définie par ses centre et voisinage, et de travailler nécessairement sur les pixels, nous allons travailler directement sur un voisinage. Nous allons pour cela nous donner un ensemble de points de référence dans notre image, appelé *Ech*, et constater si, pour chacun des points $a \in Ech$, un voisinage de a étendu sur hxl points de I possède au moins k points de la figure X traitée.

Dans les cas où cette condition se révèlera vraie, nous remplacerons le voisinage considéré par un schéma défini par avance: un segment horizontal de longueur: l pixels, et placé dans la fenêtre de centre

1. Le choix du qualificatif de ce critère provient de ce qu'un bloc de pixels, dont au moins un élément coïncide avec l'élément structurant, vérifie celui-ci: d'où la faiblesse du critère!

a et de dimension *hxl*. Nous qualifierons ce voisinage d'approché. La figure 14 donne un exemple de cette application, notée $v^{app}_{k,r}$, où *r* regroupe les paramètres *l* de largeur et *h* de hauteur pour simplifier l'écriture.

Si l'élément structurant définit la forme recherchée en morphologie mathématique, ou le noyau celle renforcée par le produit de convolution, dans le cas de notre application les paramètres *h, l, k* jouent le même rôle: la notion d'élément structurant s'est transformée en celle de forme du voisinage considéré. De plus, l'échantillonnage des points sur lesquels est évalué le nombre de points de X dans ce voisinage, diminue la complexité de l'application $v_{k,r}$, qui passe de *O(card(I) x r)* à *O(card(Ech) x r)*. Cette diminution est faible puisque le cardinal de notre ensemble d'échantillons sera important, mais révèle un avantage de notre application de voisinage approché: la résolution du voisinage varie sur plusieurs niveaux, d'abord relativement à l'échantillonnage de l'image, qui si elle atteint le niveau pixel (*Ech = I*) fait correspondre les deux applications ($v^{app}_{k,r} = v_{k,r}$), ensuite au niveau des paramètres *k, l, h* qui permettent de moduler la précision de cette application: celle-ci serait juste pour *h=1* et *k=l≠1* par exemple, au sens où elle ne détecterait que les segments de longueur *l*.

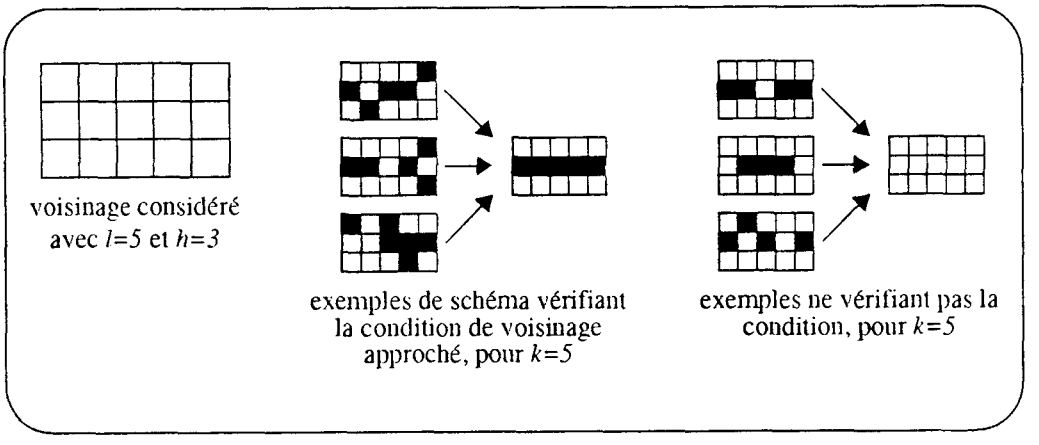


figure 14 le voisinage approché

Bien entendu, et les exemples de la figure 14 le prouvent justement -quoiqu'il soit facile d'imaginer des exemples encore plus tordus-, ce voisinage approché duquel on déduit un segment n'est pas viable localement. En effet, tout regroupement de pixels, dû au bruit, déclenchera la détection locale d'un petit segment. Mais le terme "petit" signifie bien que cette détection locale n'a rien de particulier, et qu'en fait l'on s'intéressera bien plus à la répétition linéaire de cette détection de petits segments, preuve incontournable de l'existence -probable!- d'une ligne. Nous le verrons au paragraphe suivant.

3.2.5 Simulations

Abordons maintenant la simulation de cette extraction de segments orientés. Nous passons évidemment sur le fait que seuls les segments horizontaux sont extraits et qu'il faut donc avant tout tourner l'image autant de fois de 0 à π que le veut le pas d'itération de la rotation. Voici la fonction réalisant sur une image l'extraction des segments horizontaux:

```
static void
decomp_horizon(imag)
Image        *imag;
{
int          lig, col, cpt;
```

```
Image        autrim;

Imagecpy(autrim,*imag);
for (cpt=1; cpt<=nbangle; cpt++) { /* for1 */
    for (lig = 0; lig < ((NBPIX_LIG-decalage_haut)/pave_haut); lig++)
       for (col = 0; col < ((NBPIX_COL-decalage_larg)/pave_larg); col++) {
          compose[cpt-1][lig][col] = (cassure(extraction_pave(autrim, lig*pa-
           ve_haut, col*pave_larg),seuil_info)) | | (cassure(extraction_pave
           (autrim, lig*pave_haut+decalage_haut, col*pave_larg +decala-
           ge_larg), seuil_info));
       } /* end for2 */
    rot(*imag , autrim, CTX, CTY, cpt*PI/(nbangle*1.0));
    }
}
```

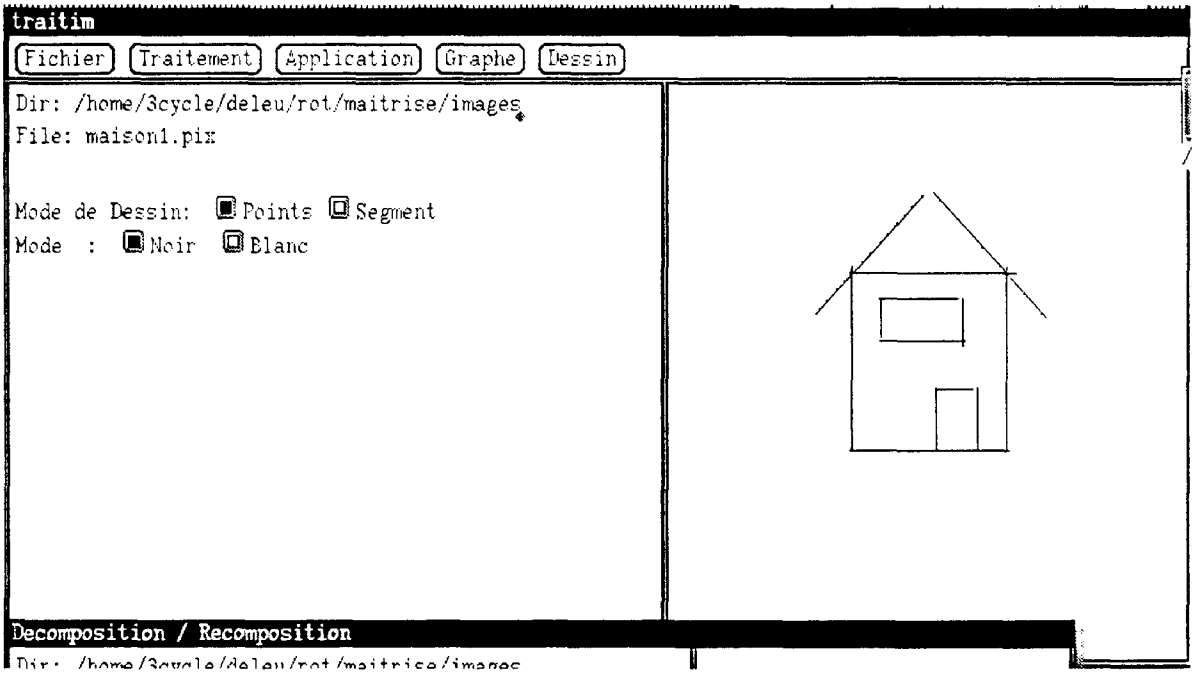
Pour chacune des images tournées successivement par la fonction rot(), la détermination des coordonnées de l'élément servant de voisinage se fait dans les deux boucles for: lig*pave_haut, col*pave_larg. La fonction extraction_pave() appelée retorune les pixels de ce pavé, de cette manière:

```
static int
*extraction_pave(im, lig, col)
Image        *im;
int           lig, col;
{
int          y, dep, compt, *pave;
Mots         *th, sh;

for (pave=Pave,y=pave_haut;y--;lig++) {
   for (compt=pave_larg,th=&(*im)[lig][col/DIV],sh=*th,dep=col%DIV;dep--;)
      sh = sh<<1;
   for (dep=DIV-col%DIV; dep-- && compt--; sh=sh<<1,pave++)
      *pave = (sh & MASK_GCHE)? 1 : 0;
   while (compt > 0)
      for (th++,sh=*th,dep = DIV; dep-- && compt--;sh=sh<<1,pave++)
            *pave = (sh & MASK_GCHE)? 1 : 0;
   }
return(Pave);
}
```

Cette fonction est un peu compliquée par le choix de coder 32 pixels sur un entier (32 bits sur les 680X0 des stations SUN). La valeur 32 est justement stockée dans DIV, pour permettre de la modifier suivant le type de machine utilisée. La donnée retournée Pave est un tableau d'entiers (1 pixel = 1 entier), que manipulera plus facilement la fonction suivante:

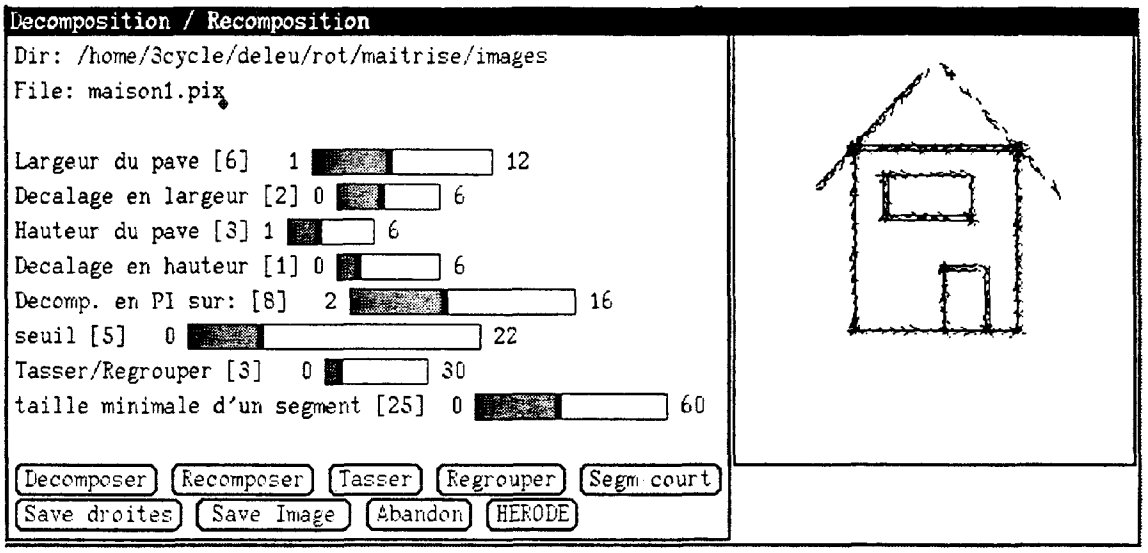
```
static int
cassure(pave, seuil)
int   *pave, seuil;
{
int             lig, col, compteur, journal;

for (journal=0, compteur=0, lig = pave_haut; lig--;)
        for (col = pave_larg; col--; )
                if (*pave++ == 1) compteur++;
if (compteur >= seuil)
journal = 1;
return(journal);
}
```

Cette fonction est très simple, puisqu'elle évalue le nombre de pixels contenus dans le pavé, le compare à un seuil, et retourne 1 ou 0 suivant ce dernier test. Voyons ce que donne visuellement de tels algorithmes:



Ceci est une copie d'écran de l'environnement mis au point pour la simulation de la machine PASTIS. Cet environnement est basé sur l'outil SUNVIEW. De là, on peut appeler plusieurs sous-environnements: traitement de l'image (rotation, homothétie, coulage), application que nous allons détailler, et le sous-environnement graphe qui servit de support à un travail dont nous reparlerons.

Voyons la décomposition en petits segments orientés de cette image. Les paramètres sont visibles, situés à gauche de l'image décomposée: le pavé a une hauteur de 3 pixels et une largeur de 6 pixels, le seuil de reconnaissance d'un segment dans un voisinage pavé est de 5, et l'image est tournée de 0 à π par pas de π/8. Voici le résultat:

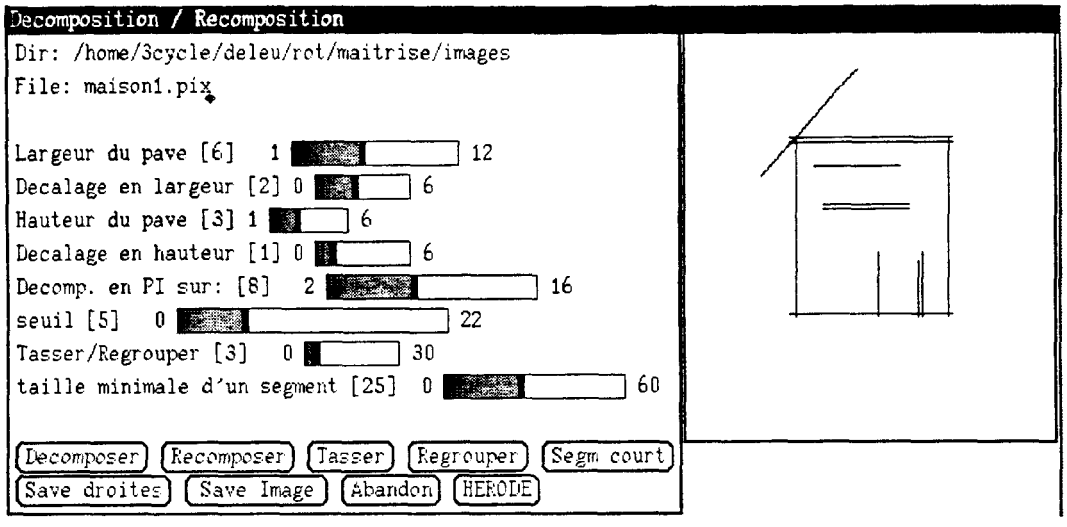


Cet exemple permet de prendre conscience de l'aspect approximatif de la démarche. Néanmoins, l'oeil est exercé et distingue déjà les regroupements de petits segments qui seront à l'origine de la détection d'une droite. Avant de passer à cette détection, il nous est apparu nécessaire de "nettoyer" l'image décomposée de ces petits segments isolés. Pour cela, l'algorithme est simple et linéaire: il s'agit de parcourir chaque image décomposée d'un certain angle, et pour chaque ligne repérer les suites de petits segments dont le nombre serait inférieur à un seuil. Voici l'algorithme:

```
while (nbanglele--) {
   for (lig = 0; lig < ((NBPIX_LIG-decalage_haut)/pave_haut); lig++)
    for (rep=0,cpt=1,col=0;col<((NBPIX_COL-decalage_larg)/pave_larg);col++)
      switch (rep) {
      case 0 : if (compose[nbanglele][lig][col]) {rep=1; cpt=1;}
               break;
      case 1 : if (compose[nbanglele][lig][col]) cpt++;
               else { rep=0;
                    if (cpt<=taillemin)
                        for(i=col-1;cpt--;i--) compose[nbanglele][lig][i]=0;
               }
               break;
      } /* end switch */
} /* end while */
```

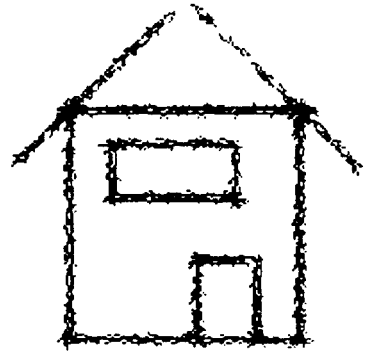
Le booléen rep permet de détecter une suite de petits segments, tandis que l'entier cpt compte leur nombre. Si ce nombre est inférieur à un seuil (cpt<=taillemin), alors ces petits segments sont détruits car considérés comme isolés.

Voici le résultat, pour un seuil de 25 pixels, c'est à dire 25/6+1=5 pavés, compte tenu de la largeur d'un pavé (6 pixels):

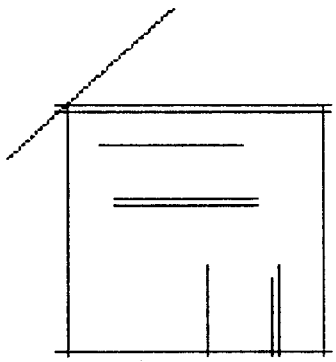


On remarque que l'image est déjà bien "nettoyée". Hélas, aucune information relative à l'existence de droites n'existe encore, mais bien plus cet exemple (choisi exprès) prouve que la discrétisation de la décomposition a fait perdre d'emblée une droite complète (à nos yeux): le pan droit du toit. D'autres petites droites se sont perdues: les côtés des fenêtres et le haut de la porte. La disparition du pan du toit provient de ce qu'il ne se trouvait pas à un angle multiple de $\pi/8$ par rapport à l'horizontal. Ce pan ne génère qu'un nombre très faible de petits segments sur le première décomposition. La disparition des petits côtés s'explique par leur taille réduite au regard du paramètre d'exclusion des petits segments isolés.

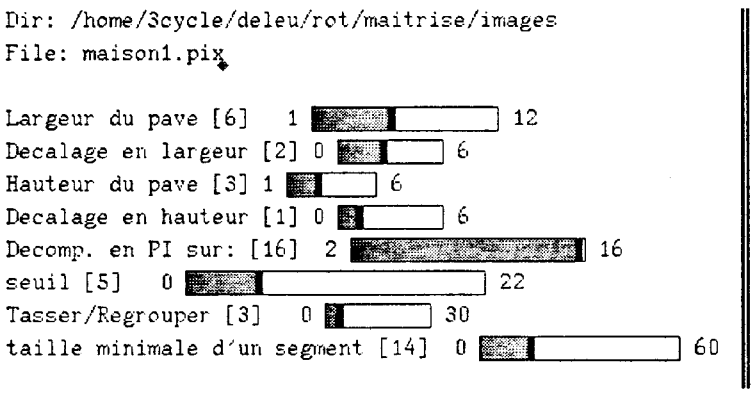
Voici une décomposition pour un pas plus fin de $\pi/16$:

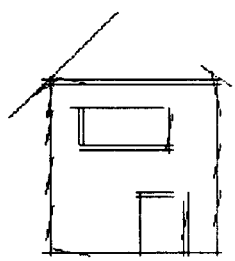


Les autres paramètres n'ayant pas évolués, on remarque tout de même que le nombre de petits segments a augmenté, et que le bas du pan droit du toit commence à apparaître. Si l'on ôte les segments isolés, voici ce que devient cette image décomposée (seuil = 5 pavés):



Le seuil d'isolement des segments n'ayant pas changé, l'image obtenue est la même que pour $\pi/8$. Par contre, si l'on modifie ce seuil:





Pour le seuil égale à 3, les côtés des fenêtres et le haut de la porte n'ont pas disparu. De même le pan droit du toit se dessine légèrement dans l'intersection avec les murs. Ceci démontre que notre méthode est certes approximative, mais peut donner des résultats de plus en plus précis par affinements successifs. L'idée serait de l'utiliser avec des paramètres "grossiers", qui permettraient de détecter les objets de taille importante, puis de modifier ces paramètres pour détecter petit à petit les objets de plus en plus fins. Les objets repérés seraient ôtés au fur et à mesure de l'image. Cette démarche fait actuellement l'objet des recherches de Michel Binse, autre thésard sur ce projet.

3.3 La détection de lignes orientées

Une fois extraits ces petits segments orientés, la phase suivante consiste à reconstruire des droites orientées. Nous présentons ici le processus de reconstruction, dont nous allons supposer qu'il ne s'effectue que sur une seule image. Cette image correspond à l'image d'origine, tournée d'un angle α et sur laquelle vient d'être appliquée la décomposition en petits segments horizontaux. Bien entendu, ce processus de reconstruction sera effectué sur chacune des images, tournée d'un angle α appartenant à $[0,\pi]$ et décomposée.

3.3.1 Détection des droites possibles

La recherche d'une droite orientée se résume à une sorte de suivi de petits segments orientés, comme il existe le suivi de contours. A la manière d'une approximation polygonale, on peut parcourir toute l'image à la recherche d'un ensemble de petits segments colinéaires, c'est à dire ici alignés horizontalement dans un plan d'angle donné. Cependant nous avons déjà insisté sur le fait qu'une telle méthode souffre du bruit ambiant propre à l'image. La perte de connexité de certains pixels lors de notre processus de rotation rapide, est en ce sens symptomatique du cas où un processus approximatif rend impossible l'exécution d'un autre processus: nous avions justement cité le suivi de contours. La méthode de suivi des segments horizontaux détectés n'est donc pas la bonne, puisque l'image a subi deux processus approximatifs successifs: rotation approchée et détection de segments horizontaux par voisinage.

Aussi, et puisque à l'époque nous ne connaissions pas la transformée de Hough, nous nous sommes tournés naïvement vers une méthode de projection sur l'axe vertical des segments horizontaux. Celle-ci consiste à parcourir l'image traitée ligne par ligne, et pour chaque ligne compter le nombre de segments horizontaux détectés. L'algorithme est extrêmement simple:

```
for (lig=0; lig<PIX_LIGNE/Pave_Haut; lig++) {
    for (cpt=0,col=0; col<PIX_COLON/Pave_Larg; col++)
        if (Compose[nb_angle][lig][col]==1) cpt++;
    Nbsegm[lig]=cpt;
```

avec Pave_Haut et Pave_Larg qui correspondent à la dimension du voisinage employé, PIX_LIGNE et PIX_COLON au nombre de pixels respectivement par ligne et par colonne. Quant à l'image décomposée, elle est stockée dans le tableau Compose[] (nb_angle spécifie l'angle de rotation de l'image). Le nombre de segments par ligne est stocké dans Nbsegm[].

Analogie avec la TH

L'angle de rotation α étant fixé, cette méthode s'apparente à l'incrémentation d'un vecteur d'accumulation de taille N, indexée à l'aide d'un seul paramètre: le numéro de la ligne parcourue. Le lecteur saisira donc l'analogie avec la transformée de Hough, où à la place d'un indice ligne on utilise un indice ρ, avec une matrice 2D qui devient elle aussi un vecteur de taille N le plus souvent pour θ fixé. L'intérêt momentané que nous accordons à cette remarque, provient du constat suivant: nous avons réussi à mettre en oeuvre une technique proche (au sens des résultats) de la TH, sans utiliser aucun multiplieur. Le seul opérateur arithmétique qui nous serve ici, est un incrémenteur. Connaissant le caractère rapide de cette méthode, et conscient de l'intérêt porté à la TH par le microcosme du traitement d'images, nous espérons que cette remarque ne tombera pas en désuétude, mais satisfera certaines curiosités.

Tandis que nous abordons l'analogie avec la TH, notons que le problème du bruit ambiant disparaît en partie grâce au "filtrage" de l'image tournée par un voisinage de type segment horizontal. Ainsi on ne travaille plus avec chaque pixel existant, mais avec des voisinages de pixels qui en regroupent plusieurs. Les pixels isolés sont donc préalablement ôtés du contenu de l'image, et ne gênent pas ainsi la détection des droites. Quant à la variabilité des paramètres qui améliore la finesse du filtrage, elle est inhérente à la décomposition en segments horizontaux puisqu'on peut aller jusqu'à un voisinage de hauteur 1 et de largeur l contenant l pixels, i.e. un segment horizontal "pur". Ainsi cette modification du filtrage par affinements successifs rejoint les idées développées actuellement sur la TH et qui concernent essentiellement l'élimination statistique du bruit.

Remarquons à ce propos les thèses de [LEVI85]:

> *"The problem of finding the four colinear points in image space was transformed into one which required the detection of a cluster of points in a parameter space. The identical result could have been achieved by scanning the image space with a line template. [...] All values of* (ρ,θ) *would have to be examined."*

Ce qu'il clôt par la remarque suivante:

> *"Thus an alternative to use the Hough Transform is to* ***rotate a template*** *about each image point in the plane. [...] All values of* (ρ,θ) *at each (x,y) in the image must be examined."*

Le lecteur notera qu'à tourner le modèle, nous avons préféré tourner l'image.

Recherche des pics

Une fois cette projection sur l'axe vertical effectuée, le résultat est un tableau de N compteurs qu'il suffit de parcourir pour en détecter les "pics". A chaque pic de ce tableau est associé une prédiction de droite possible, qui reste à confirmer. Voici l'algorithme de parcours de ce tableau et de détection des pics:

```
for (nbs=&Nbsegm[0],rep=0,ptr=0,lig=0; lig<PIX_LIGNE/Pave_Haut; lig++)
    switch (rep) {
     case 0 :   if ((nbav=*nbs++)>0)   {rep=1; Valig[ptr][0]=lig;}
                break;
     case 1 :   if ((nbap=*nbs++)<nbav)   {Valig[ptr][1]=lig-1;}
                if (nbap==0)   {Valig[ptr][2]=lig;ptr++; rep=0;}
                nbav=nbap; break;
     }
```

où la variable rep indique si un pic est actuellement atteint ou non. Notons que le test d'entrée dans cet état {nbav = *nbs++) > 0}, et le test de sortie {nbap == 0}, s'effectuent sur une valeur du tableau nulle: ceci s'explique par un pré-traitement qui a pour fonction d'ôter du tableau Compose[] les segments isolés. La valeur nulle pourrait être remplacée par un seuil de déclenchement de la détection: par exemple Nbsegm[ligne] > 10, donc on remplacerait la valeur 0 par 10.

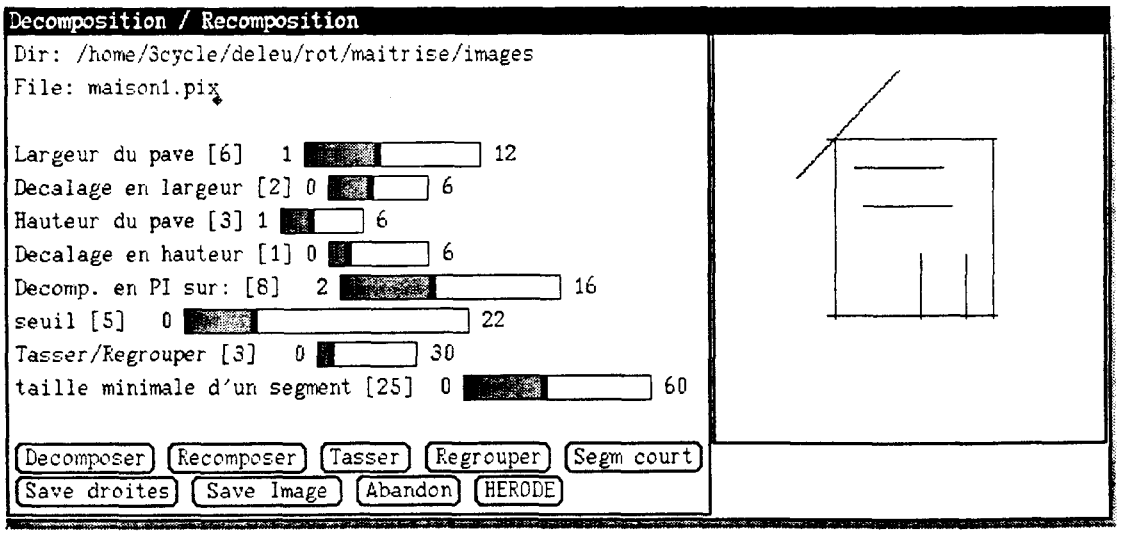
La variable ptr indique le numéro du pic détecté dans ce tableau. Comme un pic s'étale sur plusieurs lignes (cf. le terme bande horizontale), il faut stocker la première et dernière ligne de ce pic, ainsi que la ligne où la valeur Nbsegm[] est maximale. Ces données sont stockées dans le tableau Valig[ptr][]. La ligne du début du pic est dans Valig[ptr][0], la ligne de valeur maximale dans Valig[ptr][1], et la ligne de fin du pic dans Valig[ptr][2].

Un pic trop étalé n'est pas significatif, selon les approximations qui sont faites et puisque les segments isolés ont été *a priori* ôtés. Aussi un algorithme se charge de regrouper les segments qui seraient à l'origine d'un pic prononcé et étroit, en projetant tous les segments de la bande de largeur celle du pic, sur la ligne où la valeur du pic est la plus importante: numéro de ligne stocké dans Valig[ptr][1]. Les segments appartenant à des pics étalés restent en l'état. Voici l'algorithme:

```
while (ptr--)
   if   ( Valig[ptr][2] - Valig[ptr][0] < dist )
          for (rep=Valig[ptr][1], lig=Valig[ptr][0]; lig<Valig[ptr][2]+1; lig++)
                if  ( lig != rep )
                      for (col=0; col<PIX_COLON/Pave_Larg; col++) {
                            if (Compose[nb_angle][lig][col])
                                  Compose[nb_angle][rep][col]=1;
                            Compose[nb_angle][lig][col]=0;
                      }
```

Cet algorithme fait suite au précédent, ptr contient donc le nombre de pics détectés. Le test: Valig[ptr][2] - Valig[ptr][0] < dist, permet de vérifier si l'étalement de ce pic est bien inférieur à un seuil dist, avant de débuter la projection. Si tel n'est pas le cas, celle-ci n'a pas lieu et les segments ne bougent pas dans l'image décomposée. Dans le cas contraire, chaque segment appartenant à la bande de largeur: [Valig[ptr][0] , Valig[ptr][2]], est projeté sur la ligne donnée par: rep=Valig[ptr][1].

Cet algorithme effectué, notre image devient (le pas de l'angle est $\pi/8$):



L'image contient donc 9 droites, mises en évidence à l'aide du paramètre d'étalement de pics (= 3, ici).

3.3.2 Vérification de l'existence, et détermination des paramètres de la droite

La transformée de Hough est une bonne méthode de préparation à une approximation polygonale. Elle accélère l'acquisition des droites et des points de cassure, qui forment le polygone approché. Mais cette TH ne peut prétendre remplacer complètement l'approximation polygonale. En effet, une fois cette TH achevée, l'utilisateur soucieux d'obtenir une bonne description polygonale, doit venir vérifier que celle qui lui est proposée correspond au contenu de l'image, à la réalité. Les recherches effectuées actuellement sur cette TH tendent à améliorer les résultats, nous venons de le dire, et ceci justement pour éviter les descriptions trop floues, moyennes ou infidèles. Plus cette transformée sera statistiquement améliorée, meilleure sera la visibilité des pics, et donc plus rapide la phase de vérification.

Dans le même esprit, le traitement que nous proposons n'est qu'un préliminaire à l'obtention d'une description, et les détections de droites ne sont que des possibilités. Toutefois, et à la différence de la TH, l'algorithme ne donne hélas aucune indication quant à la position des points de cassure, c'est à dire les deux extrémités des droites. Aussi la seconde phase du traitement consiste à "balayer" une bande horizontale dans la zone où existe un pic.

L'utilisation d'une bande horizontale s'explique par la nécessité de regrouper entre eux les segments qui, bien que n'appartenant pas à la même ligne, sont suffisamment proches l'un de l'autre pour "fusionner" lors de la création de la droite. L'aspect approché du traitement se prolonge ici, en ce sens qu'une droite, tournée par un algorithme de rotation rapide mais imprécis, ne se transforme pas forcément en une droite aux mêmes propriétés: nombre de pixels, connexité entre eux, écart moyen entre un pixel et la droite non interpolée à laquelle il appartient,...

Quant au caractère linéaire du verbe "balayer", il indique que cette bande horizontale sera parcourue d'une extrémité de la ligne de l'image à l'autre, à la façon d'un "suivi de petits segments horizontaux". Le suivi débute au premier segment non isolé, et s'achève au dernier non isolé. Ce premier et ce dernier des petits segments détectés dans cette bande, ouvre et ferme respectivement la marche de ce suivi particulier. Ils servent d'extrémités à cette droite, et donnent par leur écartement une information approchée sur la taille de la droite.

L'algorithme qui acquiert les paramètres des droites, est alors très simple:

```
for (ang=0; ang<Nb_Angle; ang++)
   for (lig=0,col=0; lig<PIX_LIGNE/Pave_Haut; lig++,col=0)
        while (col++ < (PIX_COLON/Pave_Larg))
             if (Compose[ang][lig][col-1]) {
                  xi=col-1;
                  while (Compose[ang][lig][col] &&
                         col++<PIX_COLON/Pave_Larg);
                  xf=col;
                  stock_droite(ang, lig, xi, xf);
             }
```

Cet algorithme fait intervenir le nombre d'angles pour lequel l'image a été tournée puis décomposée. Les deux extrémités de la droite sont repérées par xi et xf, et la droite est enregistrée par la fonction: stock_droite(ang, lig, xi, xf). Remarquons que cette fonction teste d'emblée si xf-xi est supérieur à un seuil, ici 3, auquel cas elle enregistre bien ces paramètres comme appartenant à un droite détectée dans l'image.

Pour terminer, nous préciserons que cette recherche de droites s'effectue rapidement: de l'ordre d'une dizaine de secondes sur une SUN 3/80 à 4 Mips environ. Pour un programme non optimisé, bien qu'écrit en C mais perdant du temps en affichage, et traitant des images 256x256 pixels, l'ordre de grandeur de la vitesse de traitement corrobore ce que nous indiquions au début sur l'obtention d'algorithmes approchés mais rapides.

3.3.3 Perspectives et prospections

Notre travail personnel s'est arrêté à ce stade pour étudier la partie architecturale qui permettrait de réaliser rapidement ces algorithmes, bien que j'ai eu l'occasion d'encadrer des travaux ayant trait à l'intégration de l'algorithme dans un processus plus large de reconnaissance de formes. Pour situer l'algorithme présenté ici, nous dirons qu'il se situe juste en amont de l'obtention d'une description sous forme de *graphe de droites* de l'image.

Le schéma suivant situe le travail effectué dans l'optique général de ce que pourrait être un traitement complet de la machine. Les carrés identifient les objets en entrée et sortie des traitements, les hexagones représentent ces traitements:

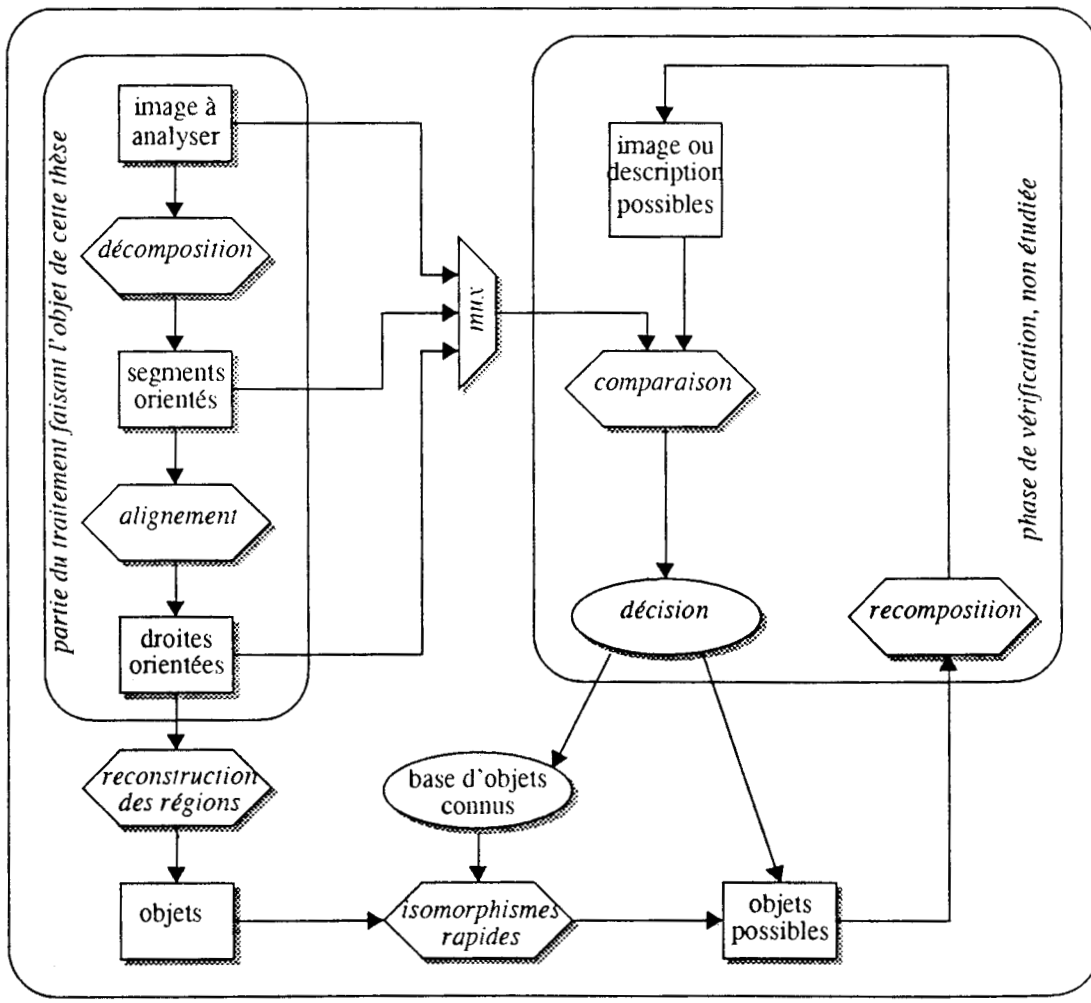


figure 15 modèle d'identification d'objets de type "prédiction - vérification"

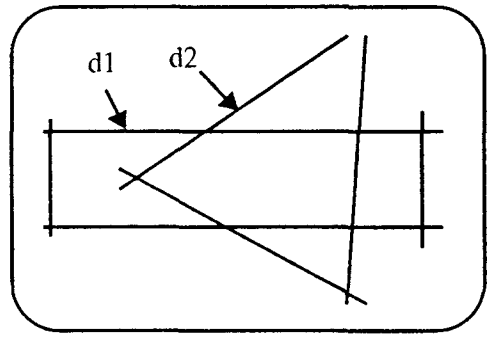
Le modèle choisi possède une stratégie d'identification des objets du type "prédiction - vérification", que l'on retrouve chez [LUX84]. Chacun des traitements doit pouvoir s'effectuer en un temps "raisonnable", c'est à dire de l'ordre du linéaire. Les figures ovales, utilisées pour la base de connaissances et le centre de décision, marquent un fait reconnu: on ne sait rien de leur organisation et fonctionnement pour le système visuel humain, et plus généralement un système de vision.

La partie gauche correspond au sujet de cette thèse. Les deux traitements auxquels nous nous sommes intéressé "de loin", sont la reconstruction de régions et les isomorphismes rapides de graphes planaires. Nous allons les décrire.

reconstruction de régions

A partir de la donnée brute que constitue une liste de droites, pour reconstituer un objet, la reconstruction de régions a pour but de comparer les extrémités des différentes droites, afin de déterminer celles qui se touchent en leur extrémité. Si une suite de droites connectées se reboucle, ces droites forment une région qui caractérisera peut-être l'objet recherché.

Ce traitement a été réalisé dans le cadre d'un projet de maîtrise au laboratoire. La difficulté se situait d'une part dans le fait que la méthode de vectorisation étant approchée, les extrémités des droites sont imprécises, d'autre part dans l'impossibilité d'utiliser les techniques efficaces pour les graphes planaires. Un graphe planaire est un graphe projeté sur un plan et dont aucun coté ne coupe un autre coté du graphe. Comme la description en segments orientés travaille par plan d'angle, il est possible d'obtenir des droites sans connaître leur intersection avec d'autres droites de cette même image: pour le plan d'angle traité, la méthode est "aveugle" vis à vis des droites existant dans un autre plan d'angle. Voici un exemple:



Dans cette figure, nous voyons deux objets bien que les extrémités des droites qui les forment ne soient pas précises. De plus, les droites *d1* et *d2* se coupent tout en appartenant à des plans d'angle différents. La construction d'un graphe planaire nécessite alors la recherche des points d'intersection de toutes les droites découvertes; ceci risque d'être coûteux, en $O(n^2)$. Si cette recherche est réalisée, on pourrait alors appliquer des algorithmes de reconstruction de régions sur graphes planaires. [SHIH89] est sur ce sujet une bonne référence, puisqu'il rappelle plus en détail les données du problème, et propose un algorithme systolique en $O(n)$, où n est le nombre de cotés du graphe planaire. Les performances de l'algorithme se base sur l'existence d'un circuit systolique de tri à $2n$ entrées, réalisant ce tri en $O(n)$ [LANG85]. D'un peu plus près, le circuit effectue un tri sur n données en $O(\sqrt{n})$; pour des données codées sur m bits, la complexité architecturale est en $O(n.m^2)$, ce qui situe la complexité[1] AT2 en $O(n^2.m^2)$.

1. A pour la surface et T pour le temps de réalisation de l'algorithme: AT^2 est un critère souvent employé pour indiquer la complexité spatio-temporelle d'un circuit.

Remarquons enfin que la notion de régions n'est pas simple, puisque si nous voyons dans la figure ci-dessus un rectangle et un triangle, ceux-ci ne correspondent aucunement aux régions qui seraient effectivement extraites dans une description en graphe planaire. Par contre, notre méthode d'extraction par plan d'angle verrait ces mêmes régions que notre oeil voit...

Isomorphismes de graphes

L'isomorphisme de graphe consiste à partir de deux graphes à vérifier leur appariement possible, *graph matching* en anglais. [BALL82] présente de manière très complète l'intérêt de cette méthode en vision, dans son chapitre 11. Il rappelle que l'isomorphisme de sous-graphes, trouver un isomorphisme entre un graphe et un sous-graphe, est un problème NP-complet, et que l'on ne sait toujours pas si l'isomorphisme de graphes est ou non NP-complet. Pour les théoriciens du domaine, [SCHO88] démontre que cet isomorphisme de graphes appartient à la hiérarchie "basse" dans les NP, ce qui voudrait dire que celui-ci n'est pas NP-complet.

Par contre, si les graphes sont planaires, des algorithmes existent qui ont un comportement polynomial. Denis Bouhineau, élève de l'ENS Lyon, a effectué un stage au laboratoire en Juin 90, afin de faire des recherches bibliographiques sur ce problème, et d'autres. S'il existe des algorithmes linéaires d'isomorphismes de graphes planaires, ceux-ci réalisent des constructions de descriptions relativement complexes, aboutissant à des algorithmes linéaires en théorie mais dont la forte constante empêche une utilisation pratique. La conclusion de [HOPC74] est à ce sujet significative:

> *"we think this planar graph isomorphism algorithm's importance is mostly theoretical, demonstrating existence rather than providing a practical algorithm"*

Une étude pratique avancée des isomorphismes sur nos graphes pourrait être un sujet de recherche intéressant. Notons qu'à ce niveau, les problèmes rencontrés appartiennent plus au domaine de l'intelligence artificielle. On les retrouve notamment dans [AI89] qui offre une synthèse des techniques d'IA appliquées à la vision.

4 Conclusion

Dans ce chapitre, nous avons découvert et approfondi la plupart des techniques relevant de la recherche d'une description sous forme de segments d'une figure. Nous avons constaté la ressemblance entre notre méthode et la transformée de Hough, tout en remarquant combien une méthode pixel pouvait se révéler plus rapide qu'une méthode numérique.

A défaut de creuser plus avant les processus à venir à la suite de cet extracteur de segments, nous allons nous intéresser dorénavant aux aspects architecturaux d'intégration de la méthode: que ce soit la conception d'un processeur de rotation rapide, puis ensuite de l'extraction de segments horizontaux de taille variable.

Références Bibliographiques du Chapitre 2

[AAIA89] Architectures Avancées pour l'Intelligence Artificielle,
11ème Journées Francophones sur l'informatique, EC2, Nancy, Janvier 1989

[AI89] Artificial Intelligence, special issue,
Artificial Intelligence, 40, 1989

[ALAG81] Alagar VS, Thiel LH,
"Algorithms for detecting m-*dimensionnal objects in* n-*dimensionnal spaces"*,
IEEE Trans. on PAMI, 3, pp.245-256

[ANN90] Artificial Neural Nets, première partie de l'International Workshop on
Algorithms and Parallel VLSI Architectures, Pont-à-Mousson, Juin 1990

[AYAC83] Ayache N,
"Un système de Vision bidimensionnelle en robotique industrielle",
Thèse Docteur Ingénieur, Paris Sud-Orsay, 1983

[BALL82] BallardD.H., Brown C.M.,
"Computer Vision",
Prentice Hall Inc., 1982

[BINS90] Binse Michel,
Systèmes Connexionnistes,
LIFL, publication interne IT178, Janvier 1990

[BROW83] Brown CM,
"Inherent bias and noise in the Hough transform",
IEEE trans. on PAMI, 5, pp.493-505, 1983

[BURN89] Burnod Y,
An adaptative Neural Network, the Cerebral Cortex,
Ed. Masson, Coll. Biologie théorique, 1989

[COHE77] Cohen M, Toussaint GT,
"On the detection of structures in noisy pictures",
Pattern Recognition 9, 1977

[DELO88] Delort François,
"Système d'approximation polygonale des contours pour application en vision industrielle",
Thèse Docteur Ingénieur, Electronique, Clermont-Ferrand 2, avril 1988

[DEUT89] Deutch J, et al,
"Performance of Warp on the DARPA image understanding architecture benchmarks",
Parallel Processing for Computer Vision and Display, Addison Wesley, 1989, Edited by Dew Earnshaw Heywood

[DUDA72] Duda R.D., Hart P.E.,
"Use of the Hough transform to detect lines and curves in pictures",
CACM 15, 1972

[FUKU88] Fukushima K,
"A neural network for visual pattern recognition",
Computer, pp.65-75, March 88 (tout le volume est dédié au connexionnisme).

[GALL86] Gallinari P, Milgram M,
"Un algorithme de suivi de contours et sa parallélisation",
Deuxième colloque image, Nice, pp.425-430, Avril 1986

[HABE82] Haber RN, Wilkinson L,
"Perceptual Components of Computer Displays",
IEEE Comp Graphics & Appl., vol. 2,No 3, pp.23-34, May 1982

[HEYD84] Von der Heydt R, et al,
"Illusory contours and cortical neuron responses",
Science, 224, pp.1260-1262, 1984

[HILL85] Hillis D,
"La Machine à Connexions" traduction de "The Connection Machine",
Masson, 1988

[HOPC74] Hopcroft JE, Wong JK,
"Linear time algorithm for isomorphism of planar graphs",
Proc. 6th Annual ACM Symp. theory of Computing, Seattle, pp.172-184, 1974

[HOUG62] Hough PVC,
"A method and means for recognizing complex patterns",
US Patent 3069654

[HUBE79] Hubel H, Wiesel T,
"Les mécanismes cérébraux de la vision",
Pour la Science, pp.79-93, novembre 1979

[HUNT88] Hunt DJ, Nolte LW, and Ruedger WH,
"Performance of the Hough Transform and its Relationship to statistical signal detection theory",
Computer Vision, Graphics, and Image Processing, 43, pp.221-238, 1988

[ILLI88] Illingworth J., Kittler J.,
"A survey of the Hough transform",
Computer Vision, Graphics, and Image Processing 44, 87-116, 1988

[JUVI84] Juvin Didier,
"Une méthode heuristique de classification et d'inspection automatique d'objets: le système de reconnaissance d'Anima",
4ème Congrès AFCET, Paris, Janvier 1984

[KUMA87] Kumar P, Krishnan V,
"Efficient image template matching on SIMD hypercube machines",
Proc. Intern. Conf. on Parallel Processing, pp.765-771, 1987

[KUNG82] Kung HT,
"Why Systolic Architectures",
Computer, pp.37-46, January 1982

[LAHA86] Lahaye JC,
Etude et réalisation d'un système de vision temps réel par reconnaissance d'éléments rectilignes,
Thèse Docteur Ingénieur, INPG (86-INPG-0129), Décembre 1986

[LANG85] LANG,
"Systolic sorting",
IEEE trans. on Computers, C-34, No 7, pp.652-658, 1985

[LEVI85] Levine MD,
Vision in Man and Machine,
Mc Graw-Hill Book Company, 1985

[LEVI85] Levine MD,
Vision in Man and Machine,
Mc-Graw-Hill Book Company, 1985

[LUX84] Lux A, Souvignier V,
"PVV, un système de vision appliquant une stratégie de prédiction-vérification"
4ème Congrès AFCET, Reconnaissance des Formes et IA, Janv 1984

[MARR82] Marr David,
Vision,
Freeman and company, NewYork, 1982

[MOUG86] Mougel H, Samy R,
"Un circuit VLSI pour la rotation d'images 2D",
2nd colloque image, CESTA, Tome 1, pp.431-436, Avril 1986

[NATO88] Natowicz René,
"Approches symbolique & connexionnistes de l'apprentissage",
Conférence Neuro-Nîmes, EC2, pp.219-238, Nov. 1988

[OLSO87] Olson TJ, Bukys L, Brown CM,
"Low level image analysis on an MIMD architecture",
1st IEEE Conf. on Computer Vision, London, 1987

[PAN90] Pan Yi, Chuang H,
"Improved mesh algorithms for straight line detection",
3rd Symposium on frontiers of Massively Parallel Computations, IEEE Comp. Press Society, Oct.1990

[PAVL82] Pavlidis T.,
"Algorithms for Graphics and Image Processing",
Springer-Verlag Berlin Heidelberg NewYork, 1982

[PRIN90] Princen J, Illingworth J, Kittler J,
"A Hierarchical Approach to Line Extraction Based on the Hough Transform",
Computer Vision, Graphics, and Image Processing, 52, pp.57-77, 1990

[RAME72] Ramer U,
"An iterative procedure for the polygonal approximation of plane curves",
Computer Graphics and Image Processing, 1, pp244-255, 1972

[RANK90] Ranka S, Sahni S,
"Image template matching on MIMD Hypercube Multicomputers",
Journal of Parallel and Distributed Computing, 10, pp.79-84, 1990

[RESN89] Resnikoff HL,
The Illusion of Reality,
Springer-Verlag NewYork, 1989

[ROSE88] Rosenfeld A, Ornelas J, Hung Y,
"Hough transform algorithms for mesh-connected SIMD parallel processors",
Computer Vision, Graphics, and Image Processing, 41, pp.293-305, 1988

[SCHO88] Schoning Uwe,
"Graph isomorphism is in the low hierarchy",
Journal of Computer and System Sciences, 37, pp.312-323, 1988

[SERR86] Serra Jean,
"Introduction to Mathematical Morphology",
Computer Vision, Graphics and Image Proc., 35, pp.283-305, 1986

[SHIH89] SHIH Zen-Chung, et al,
"A systolic algorithm for extracting regions from a planar graph",
Computer Vision, graphics, and image proc, 47, pp.227-242, 1989

[SHNE81] Shneier M,
"Two hierarchical linear features representations: Edge pyramids and edge quadtrees",
Computer Graphics Image Processing, 17 pp.211-224, 1981

[SILB85] Silberberg TM,
"The Hough transform on the Geometric Arithmetic Parallel Processor",
Proc. of the IEEE Workshop on Computer Architecture for Pattern Analysis and DB managt, Dec. 1985

[THOR88] Thorpe Sj,
"Traitements d'images chez l'homme",
TSI, vol. 7, No 6, pp.517-525, 1988

[VEEN81] Van Veen TM, Groen FC,
"Discretization errors in the Hough transform",
Pattern Recognition, 14, pp.137-145, 1981

[WALT87] Walters D,
"Selection of Image Primitives for general-purpose Visual Processing",
Computer Vision, Graphics, and Image Proc., 37, pp.261-298, 1987

chapitre 3

Les Modules Architecturaux Dédiés

Cette dernière partie concerne l'architecture de la machine Pastis. Elle comporte une réflexion sur l'architecture de cette machine pour faire de la vision. Nous proposons un schéma simple pour une machine d'évaluation, qui permettrait d'étudier les différents algorithmes. Ensuite, nous donnons un schéma pipeline pour une machine dédiée au fonctionnement intense.

Ensuite, les problèmes plus particuliers d'architecture qui furent le centre d'intérêt et le coeur de cette thèse sont présentés. D'abord, l'architecture du processeur de rotation rapide est décrite sur trois plans: l'existant dans ce domaine, une description architecturale et une description fonctionnelle. Les détails de réalisation du processeur sont accessibles dans une publication interne du laboratoire [DELE91]. On étudie dans une troisième partie l'extraction de segments ainsi que la phase de vectorisation. La gestion de la mémoire d'image, et les techniques proposées qui restent à évaluer, achèvent ce chapitre.

chapitre 3

Les Modules Architecturaux Dédiés

1 Architecture de la machine Pastis

Après la description de la méthode d'extraction de segments, et la confirmation visuelle empirique de son efficacité, nous pouvons aborder le côté architectural de notre étude. Celui-ci ne saurait débuter sans un rapide aperçu des idées directrices qui nous ont amené au concept MAD, c'est à dire Module Architectural Dédié.

1.1 Idées directrices

Quel modèle?

Au chapitre 1, nous avons présenté les différents modèles employés pour les machines de vision. Si l'architecture SIMD, et celles qui en dérivent, dominent cette présentation tant par leur originalité, leur adaptabilité et le rapport étroit qu'elles entretiennent avec l'intuition que nous avons tous du calcul parallèle, elles possèdent néanmoins deux défauts majeurs qui sont le coût d'une machine basée sur ce modèle et le principe de projection d'un pixel sur un processeur élémentaire.

La solution à cette projection est d'ordre logiciel. L'émulation d'une machine virtuelle SIMD tableau de N^2 processeurs se réalise à l'aide d'une machine réelle SIMD tableau de n^2 processeurs. Les conséquences sur les performances dépendent alors non seulement du rapport N/n, mais bien plus du potentiel d'entrée-sorties de la machine réelle (cf. le registre S de transfert sur la machine MPP). Le modèle systolique est en ce domaine une référence.

Sur la question du coût, nous ne nous étendrons pas, mais nous constatons que le coût nous oblige à nous servir des machines que font et savent faire mieux que nous d'autres personnes; ce qui ne saurait être mauvais en soi. Toutefois l'économique n'est pas le seul épithète, puisque la notion de coût rassemble pêle-mêle un certain nombre de paramètres complexes: coût économique, limite technologique, limite des communications,... En fait cet coïncidence des paramètres ne permet pas de développer une machine effectivement adaptée aux besoins des algorithmes: notamment un nombre de processeurs du même ordre que les 10000x10000 pixels que traite aisément l'oeil, autour d'un réseau de communications performant.

Toutefois les progrès réalisés dans le domaine de la fabrication d'ASIC, pourraient à terme inverser cette tendance, en autorisant la création de machines dont on ne saurait utiliser pleinement les possibilités. En effet, les traitements d'image utilisent un mode de communication locale dont on sait qu'il se projette facilement sur une architecture 2D. Le problème essentiel est à ce niveau d'intégration de calculer non seulement la performance des algorithmes, mais surtout leur temps de chargement. L'aspect pipeline du cortex visuel répond justement à cette contrainte.

Face aux réalisations SIMD, les architectures MIMD s'insèrent le plus souvent dans les phases de reconnaissance, d'analyse *intelligente* du contenu de l'image. Nous avons vu que certains algorithmes développés sur machines MIMD s'avèrent bien moins efficaces que leur semblable sur machines SIMD. Au niveau des pixels ou aux prémices d'une description, les traitements sont localisés et identiques, et la quantité d'information gérée trop importante pour permettre des mécanismes sophistiqués de partage de données. Un réseau à entrelacement n'est pas aussi efficace qu'une machine tableau pour des échanges locaux. Par contre, cette architecture à processeurs multiples, distincts et dédiés présente de nombreux avantages:

(i) coût relativement faible de chacun des processeurs,
(ii) puissance importante des processeurs puisqu'ils sont dédiés,
(iii) stratégie globale de gestion des processus facilitée.

De plus, par machine MIMD l'on ne pense pas uniquement aux machines multi-microprocesseurs. Certaines d'entre elles fonctionnent avec un amalgame hétéroclite de processeurs dédiés, de microprocesseurs, de DSP et de plans mémoire.

La technique de l'ASIC s'impose donc d'elle-même, puisqu'elle permet à la fois d'adopter une orientation SIMD en interne dans ces circuits, et une architecture MIMD entre les circuits ou quelques groupes de circuits. La construction qui s'ensuit tient à la fois des modèles pyramidal, systolique, ou plus simplement de l'aspect pipeline des opérateurs de traitement. Ce pipeline est d'ailleurs le paradigme en matière de vision humaine, pour ce qu'on sait du cerveau.

MAD ou DSP?

Le coût d'un processeur ASIC n'est pas rédhibitoire, surtout pour un projet de recherche. La taille du processeur est ici essentielle, et l'optique de fabrication d'un prototype réduit semble la bonne démarche. La difficulté de l'approche *architecture dédiée* est d'en tirer un réel avantage. Beaucoup de processeurs puissants sont utilisés en traitement d'images, tels que le *i860* d'*Intel*, qui suffisent amplement au besoin. Avant de proposer la fabrication d'un processeur dédié, il faut donc argumenter en sa faveur, face aux processeurs à usage général ou DSP.

L'efficacité de la gestion globale du processus de vision est l'un des arguments en faveur du MAD. Si l'on se place au niveau *tâche*, un processeur dédié effectue une tâche précise en un temps connu, ce processus nécessite une quantité d'information finie. Comme les processeurs sont puissants et donc le temps d'exécution d'un processus court, on peut envisager de bloquer l'accès aux données demandées par un processeur. Concernant certains traitements, cette approche n'est pas concevable avec des processeurs de type DSP, qui comme nous le verrons pour la rotation ne peuvent effectuer en un temps très court (de l'ordre de la milli-seconde) des tâches simples sur un million de pixels.

Par contre, et à la manière de ce que propose CAPITAN, une cohabitation de MAD et de DSP est une bonne solution. Certains traitements n'ont aucunement besoin d'une réalisation ASIC pour s'exécuter rapidement. La contrainte qui pèse sur les algorithmes est d'avoir un comportement linéaire, comme nous le verrons pour la phase de détection de segments.

PASTIS, machine d'évaluation

L'approche du projet PASTIS 90 est double: proposer des algorithmes approximatifs originaux, fonctionnant *quasi*-linéairement quand ceci est possible, et concevoir des circuits adaptés à la réalisation de ces algorithmes. Aucun modèle n'est pratiquement retenu d'emblée; nous retenons les méthodes existantes les plus adaptées à nos besoins. Cependant deux hypothèses sont posées dès le début:

- les processeurs réalisés ne sont que des prototypes pour évaluer la faisabilité de la machine, et les techniques utilisées pour leur conception sont donc les plus simples: ainsi le dessin des ASICs se fait en *standard cells*, aux dépens des possibilités offertes par l'intégration *full-custom* (vitesse, meilleure intégration),

- la machine est construite sur la structure la plus simple et la plus adaptable possible, bien qu'elle ait à supporter des débits d'entrée-sorties importants qui correspondent aux transferts d'images complètes entre les processeurs.

Ces remarques nous amènent évidemment à retenir un modèle de machine proche de TOSPICS et d'autres unités de traitements d'image présentes dans le commerce. Le paragraphe suivant expose brièvement la structure de PASTIS.

1.2 Architecture globale de Pastis

Comme l'indique la figure 1, notre proposition pour la machine Pastis est une architecture à partage de données entre des Modules Architecturaux Dédiés (MAD), travaillant en parallèle. En l'état actuel du projet, ces données partagées sont de deux types: des images et des graphes planaires. Le partage est assuré physiquement par l'existence de deux bus distincts.

1.2.1 Communication inter-MADs

Le choix du partage des données *via* un bus tient à deux remarques. La première est pragmatique, et concerne le coût de développement lié à l'utilisation de bus. En effet, le partage de données dans un environnement multi-processeurs (ici les MADs) peut se faire de différentes façons, présentées dans la première partie de cette thèse: bus, *crossbar*, réseau Oméga... Parmi ces techniques, le bus apparaît la méthode la moins coûteuse, mais aussi la moins performante. Ce désavantage est compensé par l'aspect machine ouverte qu'apporte un bus.

L'ajout de processeurs ne nécessite aucune refonte globale de l'architecture, et c'est là la deuxième raison du choix du bus. Effectivement, PASTIS 90 est avant tout une machine d'évaluation de techniques originales d'analyse d'images et de reconnaissance de formes. Elle se doit donc d'offrir à l'utilisateur la plus grande souplesse d'adaptation à la mise en oeuvre de nouveaux modules dédiés dans son environnement. Dès lors, le bus est de loin la technique adaptée à ce type d'architecture ouverte; son utilisation largement répandue dans de nombreux systèmes évolutifs de vision corrobore notre choix.

Les Unités de Contrôles (UC) interviennent aux deux niveaux de la machine: images ou graphes, pour éviter les situations critiques d'accès aux données. Les UC sont programmées pour répondre en un temps borné aux demandes de la machine hôte, qui gère la distribution des tâches. Chaque UC contrôle l'initialisation des tâches qu'effectuent ensuite les MADs. De plus, les UC veillent à l'ordonnancement des tâches afin d'éviter l'exécution en parallèle de processus se déroulant sur une même partie de la mémoire. Ce travail est étroitement lié au *mapping* de la mémoire d'images dans le cas d'une mémoire à bancs séparés. Chaque UC a donc comme tâche de gérer la distribution des images dans la mémoire, de contrôler l'ordonnancement des tâches et de les initialiser.

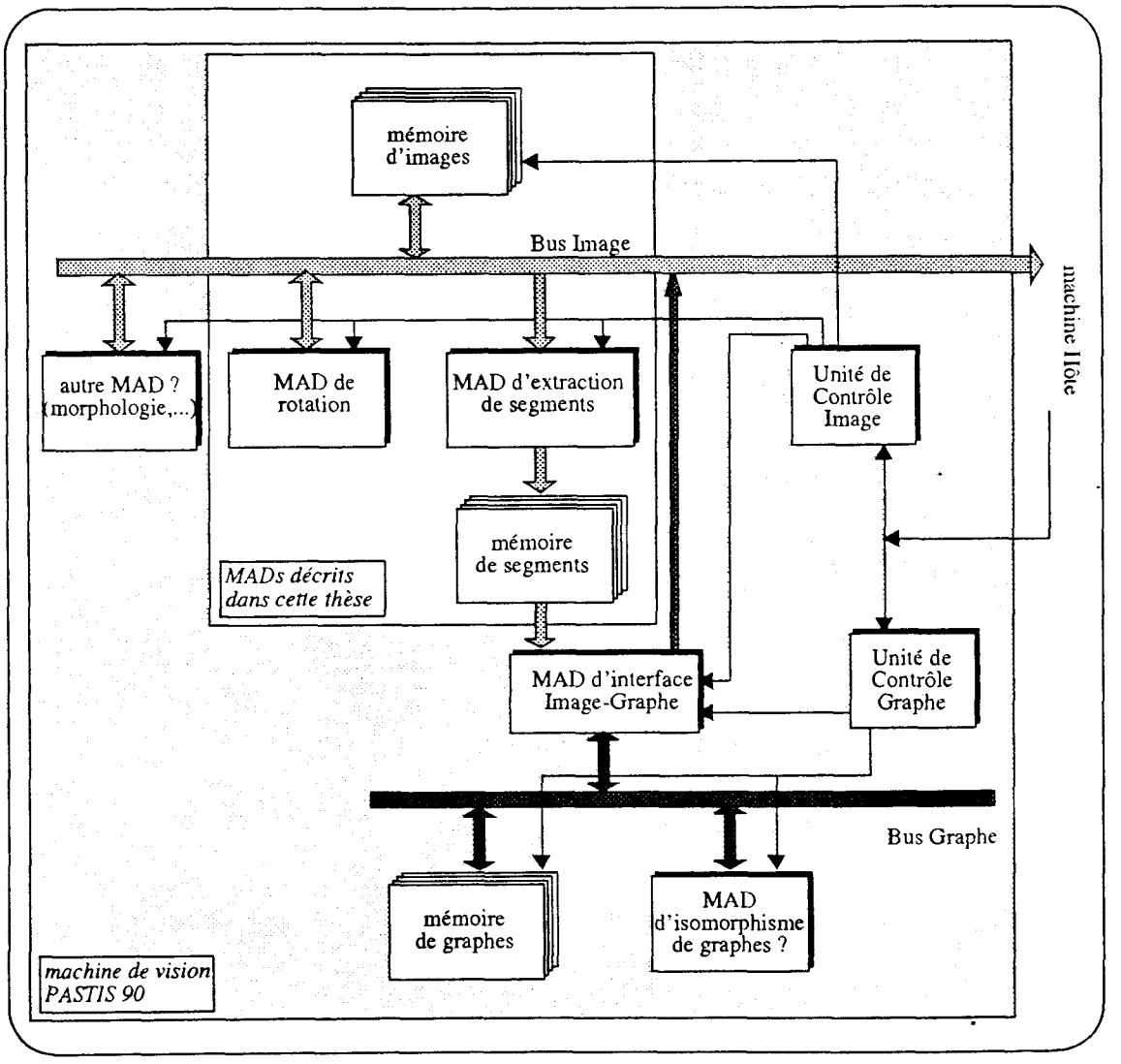


figure 1 Architecture globale de PASTIS 90

1.2.2 Architecture des MADs

Quant aux MADs eux-mêmes, il s'agit de processeurs *a priori* indépendants, qui se partagent la maîtrise de l'accès au bus de leur niveau, et exécutant pendant un temps donné la tâche que l'UC leur a confiée. Le temps connu d'exécution d'une tâche est la limite supérieure que peut évaluer l'UC à l'appel du MAD. Il dépend des paramètres suivant:

(i) temps d'exécution, par le MAD requis, d'une tâche élémentaire sur un mot (sa taille dépend du MAD),

(ii) temps d'accès via le bus à l'ensemble des mots traités par cette tâche élémentaire (lecture, écriture),

(iii) nombre de tâches élémentaires par tâche effectuée,

et enfin, les paramètres dus à l'environnement multi-processeurs:

(iv) mode de fonctionnement du bus de partage des données (*burst*[1] ou normal),

(v) charge du bus (requêtes effectuées par les MADs actifs).

Nous observons que la capacité de notre machine est étroitement liée aux performances concurrentes des MADs et du mécanisme de partage des données.

Le déclenchement d'une tâche par l'UC se fait par interruption du MAD concerné. Pour éviter d'interrompre un MAD actif, l'UC dispose d'un bit d'état de chacun des MADs qu'elle contrôle. Ce bit d'état renseigne sur l'état actif ou non du MAD.

Par programmation de l'UC, il serait relativement aisée de développer une machine temps réel. L'UC affecterait une priorité aux tâches des MADs, et déciderait en fonction de ce paramètre de l'interruption ou non de l'exécution d'une tâche au profit d'une autre. Nous n'avons pas retenu cette possibilité, toujours pour une raison de simplicité matérielle et logicielle, synonyme de coûts moins élevés de développement. Néanmoins, dans le cas où cette solution serait retenue, la connaissance d'une borne supérieure sur le temps de traitement étant un point (acquis) important des machines temps réel, les MADs devraient être pourvus d'un mode de sauvegarde de leur état (conservation de contexte). Le plus difficile dans cette configuration serait de garantir l'inviolabilité des données traitées par la tâche interrompue.

Pour l'architecture des MADs eux-mêmes, deux solutions sont envisageables qui sont liées aux performances attendues. Dans le cas de traitements relativement courts, c'est à dire ne nécessitant pas une puissance de calcul importante, une carte CPU restreinte est une solution fiable. Cette carte contient un processeur du commerce, qui possède sa propre mémoire de travail (codes et données), un accès au bus et un mécanisme d'interruptions pour les requêtes par l'UC de son niveau. Ces cartes ont une grande

1. le mode burst permet le transfert de plusieurs mots *via* le bus à chaque acquisition du bus par un maître: le maître du bus ne donne qu'une adresse (celle de départ) et écrit ou lit les données consécutives à cette adresse sans que celle-ci ne soit modifiée. Ce mode est particulièrement intéressant dans les bus multiplexés (adresse/donnée), c'est à dire ceux de taille importante comme le nôtre.

capacité d'adaptation à différentes classes de traitements, par programmation. L'utilisation d'un circuit DSP récent serait alors un atout pour son architecture sophistiquée: noyau RISC, plusieurs unités fonctionnelles, et plusieurs instructions par cycle. Néanmoins il convient de souligner que notre objectif n'est pas la réalisation d'une machine conventionnelle utilisée en vision, et qu'un tel choix dénaturerait l'intérêt de notre exposé.

La deuxième solution, bien plus exigeante, implique la réalisation de modules réellement dédiés à une classe de tâches, quand ce n'est à une seule tâche. Nous avons débuté cette réalisation pour le processeur de rotation, et étudié d'autres éléments: processeurs d'extraction de segments, de vectorisation, gestion de la mémoire. L'apport des techniques VLSI est pour nous fondamental, puisqu'il nous permet de transférer des techniques éprouvées de façon expérimentale par logiciel vers une implantation matérielle. Il nous oblige donc à cerner de près des problèmes de faisabilité et de temps de réponse trop souvent occultés par la phase logicielle.

L'architecture des circuits dédiés doit permettre aux processeurs de répondre à des critères très contraignants de rapidité de traitement. Pour cela, le double parallélisme: spatial (N processeurs) et temporel (pipeline) devient indispensable. Il induit donc certaines contraintes tant dans la conception de micro-architectures: circuits réalisant un traitement donné pour une classe de tâches délimitée, que dans la conception macro-architecturale: mise en parallèle de MADs distincts fonctionnant simultanément.

Pour fondre des circuits dédiés, nous recourons aux circuits intégrés pour les applications spécifiques (ASIC). Ces circuits ne possèdent pas les capacités de contrôle et d'enchaînement d'actions des microprocesseurs. Chaque MAD est donc composé d'un ensemble de circuits dédiés et d'une unité de séquencement simple qui les contrôle. Ce séquenceur peut être soit câblé soit micro-programmé. Nous retiendrons la solution micro-programmée, seule à offrir la *re*-programmation des MADs par les utilisateurs. Elle présente l'avantage d'être facile à mettre en oeuvre, car le code est d'autant plus simple que les fonctions réalisées par chacun des MADs sont *câblées* par définition, et qu'il suffit de contrôler l'enchaînement de ces fonctions. Cette *re*-programmation importe en effet beaucoup dans une machine d'évaluation *(cf. figure 2)*.

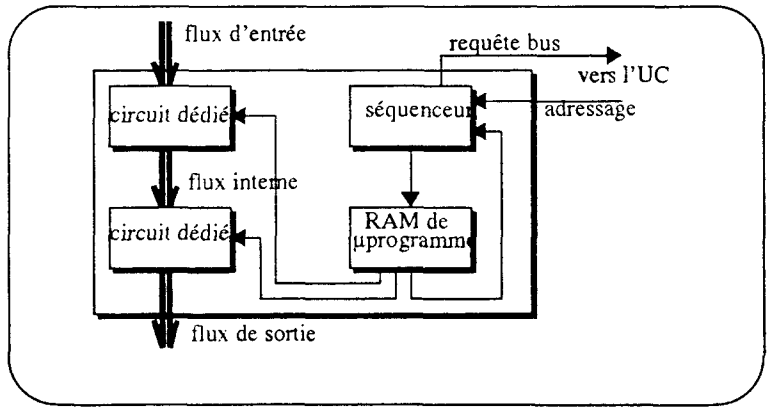


figure 2 Schéma synoptique d'un MAD micro-programmable

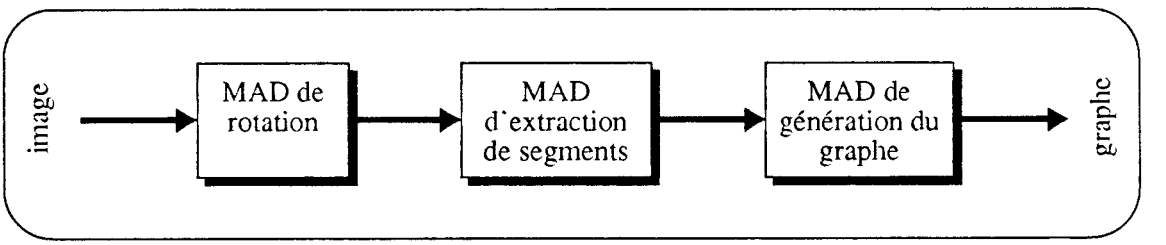
Ce synoptique est général, et ne prend pas en compte par exemple un déroulement conditionnel du flot de µinstructions généré par le séquenceur. Les flots d'entrées-sorties auront une place importante de part l'aspect systolique des circuits dédiés utilisés, comparés au flot de µinstructions réduit à un simple rôle de contrôle des circuits.

1.3 Prolongement vers une machine temps réel

Si le schéma de la figure 1 s'appuie sur un partage des données par un bus, le pipeline *{mémoire d'images, MAD d'extraction de segments, mémoire de segments, MAD d'interface Image-Graphe, mémoire de graphes}* démontre que notre machine accueille des processeurs conçus pour des traitements "en chaîne", sans influencer la conception de ceux-ci. Cette structure d'accueil devrait pouvoir être à tout moment revue, repensée, pour offrir une machine de vision réellement temps réel, et non limitée par les débits de son support matériel.

Pour cela, la conception des processeurs dans une optique systolique[1] offre la possibilité de les intégrer dans une structure d'accueil copiée du schéma proposé au chapitre 2 à la figure 15. Ce schéma se retrouve aussi chez Thorpe [THOR88, page 522], dont nous avons précisé les travaux au paragraphe 3 de ce précédent chapitre. Il montre la décomposition en aires du cortex visuel, telle que la conçoivent les neurophysiologues à la lueur de leurs études les plus récentes. Chaque aire visuelle correspond à une sorte de processeur indépendant, au parallélisme massif; certaines d'entre elles, les aires V1 et V2 possèdent une composition en couches, où chaque couche contient un ensemble de neurones qui est par exemple plus sensible à la couleur tandis que sa couche voisine est plus sensible à l'orientation, et ainsi de suite... Cet entrelacement correspond de fait à des processus indépendants; tous les processus attachés à la réalisation d'un traitement donné, regroupent en sortie leur résultat afin de les envoyer à l'aire suivante.

On retrouverait un schéma de ce type:



Nous avons donc voulu, pour le processeur de rotation notamment, concevoir un modèle qui puisse prendre en compte cette particularité que serait la mise en pipeline des traitements dans notre machine. Pour le processeur de rotation, nous allons voir que cette volonté s'est réalisée malgré un dessin complètement parallèle du circuit.

1. ce terme semble caractériser au mieux nos processeurs dans leur architecture interne: parallélisme massif, communications locales, opérations synchrones; tout en incluant l'arrangement externe de ceux-ci: mise en pipeline des processeurs, débits E/S importants,... L'adjectif pipeline serait à nos yeux trop réducteur.

2 Le MAD de Rotation

La rotation d'images est un processus qui intervient souvent dans les applications de poursuite de cibles et guidage d'engins, de vision médicale, de synthèse d'images,... L'inconvénient de ce processus provient du nombre de calculs qu'il requiert, et du nombre élevé d'entrée-sorties.

Dans le cadre du traitement de l'image, nous avons imaginé de réduire la complexité de la rotation, en approchant celle-ci, afin de contrebalancer cette perte de précision par un gain en rapidité. Voici les aspects matériels dérivés.

2.1 Introduction

La décomposition d'images en segments orientés de taille fixe, et les comparaisons rapides, à une similitude près, nécessitent la conception d'un *opérateur de rotation rapide* et son intégration dans la machine PASTIS. Nous allons décrire la façon dont s'obtient cette rapidité aux dépens de la précision; l'image tournée peut ainsi perdre tout ou partie de la connexité de ses pixels.

La conception d'un tel opérateur de rotation conditionne la réalisation d'une machine viable; ceci est d'autant plus important que notre algorithme de vectorisation impose un nombre élevé de rotations[1], et que la rotation reste longue à mettre en oeuvre actuellement. Dans un contexte d'utilisation intense, et d'une architecture classique à base de DSP, PASTIS 90 ne supporterait pas le coût d'une rotation précise sur une image de 512x512 pixels. Ces remarques ont fait du processeur de rotation rapide le premier circuit réellement développé par notre groupe.

La démarche du groupe est décrite dans la suite de ce paragraphe: recherche d'un parallélisme dans l'algorithme de rotation, simulations de cet algorithme, et enfin, définition globale d'une architecture répondant aux critères de Pastis: approximation, rapidité, orientation temps réel, parallélisme intense, à la fois spatial et temporel. En cela, la lecture de ce paragraphe permet de s'imprégner de ce que nous avons déjà référencé comme la *philosophie* du projet PASTIS.

2.1.1 Utilisation de la rotation précise

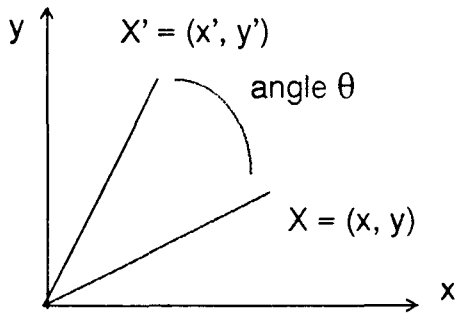
Une rotation précise[2], de centre X_0 (x_0,y_0), est l'application à chacun des pixels de l'image de la transformée d'équation suivante:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \times \begin{bmatrix} x - x_0 \\ y - y_0 \end{bmatrix} + \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$$

1. Que ce nombre soit en réalité de trois, voire quatre, importe peu en fait. La rapidité du processus de rotation est un élément clef de notre vectorisation, car il permet d'en diminuer la complexité comme nous l'avons vu au chapitre précédent. Notons que le nombre de rotations croît linéairement avec la précision de cette vectorisation.

2. la rotation précise est la rotation 2D habituelle, par opposition à notre méthode qui aboutit à une rotation imprécise.

Le point X = (x, y) est transformé en X' = (x', y') comme suit:



Pour chaque point (il y a 256 K pixels dans une image 512x512), il faut donc effectuer 4 multiplications et 2 additions[1], suivies de l'interpolation entière des coordonnées de X'. Le processus de rotation nécessite à peu près un million d'opérations, dans le cas de l'interpolation au plus proche voisin. La taille de l'image et la précision attendue sont deux paramètres qui évitent d'utiliser des calculs flottants, toujours plus long du fait des normalisations. Notons que pour une image 10000x10000, ce nombre d'opérations serait donc de l'ordre de 400 millions... Si la rotation doit être obtenue en 1 milli-seconde, la puissance attendue est de l'ordre des 0,4 *Téra* instructions par seconde...

Cependant, le domaine de traitement de PASTIS n'est pas celui des images réelles: dans le dessin au trait, la quantité de pixels à tourner est bien inférieure à celle donnée ci-dessus. De plus, il s'agit de pixels noirs ou blancs, donc codés sur 1 bit (1 ou 0) et non sur 8 bits, le niveau de gris. En moyenne, on estime leur quantité à 1% du nombre total de pixels. Il resterait donc 2500 pixels à tourner, dispersés aléatoirement dans la mémoire d'image; dispersion qui ne saurait faciliter leur accès en lecture ou en écriture.

Dans cette hypothèse, un processeur de calcul rapide peut être retenu. Nous pensons notamment à des *processeurs DSP* (*Digital Signal Processing*), très rapides et désormais dotés d'unité de calcul en virgule flottante et de mémoire RAM *on-chip*. Par exemple, pour n'en citer qu'un, le T9506 de la firme *Toshiba*, qui travaille à une fréquence de 5 MHz, possède une structure interne à trois étages de pipeline. Il est capable de tourner un pixel en 200 ns. Pour l'ensemble des pixels on l'obtient en 0,5 ms, si l'on est capable de fournir les coordonnées des pixels noirs à cette allure, sans oublier de les ranger après rotation. Nous verrons d'ailleurs que la rotation est non seulement un problème difficile du point de vue du calcul, mais aussi du point de vue du nombre d'entrée-sorties générées.

Bien que pour le dessin au trait une approche *DSP* soit viable et peu onéreuse, la recherche d'un fonctionnement en temps réel de la machine PASTIS souffre de l'indéterminisme du temps de réponse. Le temps réel suppose la connaissance *a priori* du temps de réponse, et l'évolution dans les meilleurs cas et non dans des cas moyens (en terme de temps de réponse, quantité de traitements,...). C'est ainsi que nous nous sommes tournés vers les machines parallèles effectuant la rotation rapide 2D.

1. La rotation incrémentale, que nous décrivons par la suite, permet de limiter à 2 additions réelles le nombre des opérations. Il n'en reste pas moins que les interpolations en demandent beaucoup.

2.1.2 Architectures Parallèles

Nous nous intéressons ici aux développements du processus de rotation 2D, soit sur machines parallèles soit par des circuits VLSI dédiés. Bien que cette partie fasse office d'état de l'art, elle permet aussi de prendre conscience des particularités propres à chacune des approches parallèles de la rotation.

Architecture SIMD

Le processus de rotation peut s'effectuer sur une machine SIMD. Le processus de rotation distribue les pixels à tourner sur les PEs; les PEs calculent la position dans l'image tournée de leurs pixels. La machine de rotation, proposée par Arabnia [ARAB87-1], est conçue pour une telle architecture. Elle a d'ailleurs été testée sur la machine DAP d'ICL, offrant des temps de rotation faibles: entre *40* et *90 ms* pour des images binaires.

Actuellement à l'étude dans le cadre du PRC-AMN [PRC89], la machine MASIVE devrait effectuer une rotation d'image 512x512, en temps constant: *20 ms* (prévision). La technique est la même: chacun des PEs calcule la nouvelle adresse des pixels saisis. Le coût est cependant important: il s'agit d'une machine complète de traitement d'images, à base de processeurs 1 bit, d'architecture proche de celle du GAPP. Ces algorithmes n'ont rien de sensationnels, et n'existent que du fait de la machine elle-même. Nous n'y reviendrons donc pas.

Architecture MIMD

Arabnia propose une autre architecture, à base de transputers [ARAB87-2]. L'image est divisée en M blocs de P lignes, et chaque bloc est réparti sur un des M transputers. Le contenu de l'image est décomposé en **stripcode**: un segment de longueur variable (premier paramètre) constitué de pixels de même couleur (second paramètre). Chacun des M blocs contient donc un certain nombre de stripcodes.

La rotation s'effectue en deux étapes:

(i) Chaque transputer calcule les stripcodes tournés d'un angle α, de son bloc. Puis, les segments obtenus sont échantillonnés, afin d'être affichés point par point sur un écran.

(ii) Les stripcodes obtenus traversent alors un anneau de communication entre transputers, et ceux-ci prennent les stripcodes les concernant. La rotation est terminée quand l'anneau ne transporte plus de points.

Le choix de la topologie du réseau est décisif pour les performances attendues. Remarquons toutefois que cette machine n'a pas un temps de réponse constant, ou connu *a priori*. Celui-ci dépend du nombre de stripcodes contenus dans chacun des blocs (la borne supérieure est obtenue pour un damier, par exemple), ainsi que de leur disposition qui influent sur la charge du réseau de communication (borne probabiliste uniquement, difficile à évaluer). Cette remarque est générale aux machines du type MIMD auxquelles celle-ci se rattache.

Dans l'architecture MIMD, chaque processeur possède sa propre unité de contrôle. Il est donc capable d'exécuter son propre programme sans synchronisation. Les communications entre processeurs sont alors primordiales, mais aléatoires dans le temps et dans l'espace.

Ces processeurs ne font pas généralement l'objet d'études spécifiques. Il s'agit de processeurs existants, non-dédiés mais supportant un taux élevé de communications (e.g. Transputer), voire de changements de contexte (Processeurs RISC, comme le 80960). Quant aux accès aux images d'entrée et de sortie, une mémoire entrelacée en entrée peut être utilisée. Hélas on ne peut garantir à tous les coups un accès "symétrique" de la mémoire de sortie: la rotation est en cela un mauvais algorithme.

La réalisation d'Arabnia [ARAB87-2] souffre de ces contraintes, malgré une topologie du réseau de communication et un codage de l'information qui apparaissent intéressants. Pour des images réelles, une rotation s'effectue en 100 à 500 ms, suivant le nombre de transputers et de stripcodes.

Processeurs VLSI de rotation

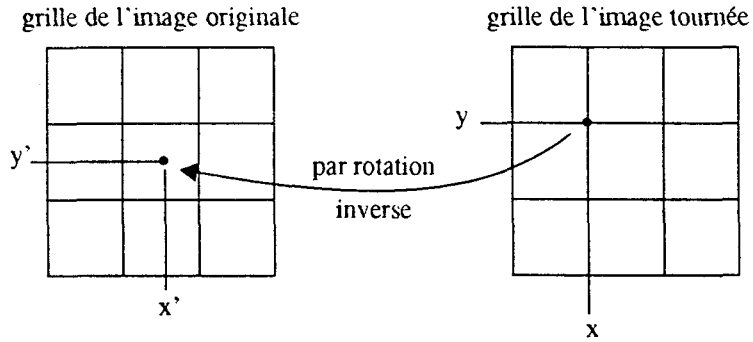
[MOUG86] présente un circuit VLSI pour la rotation d'images 2D. Cette rotation est précise, et les auteurs se préoccupent essentiellement de la fonction d'interpolation. Cette fonction est appliquée aux coordonnées réelles obtenues par les équations de la rotation 2D, pour calculer les coordonnées entières correspondant à la position du pixel atteint. En effet, toute transformation géométrique se décompose en deux étapes:

(i) le calcul des nouvelles coordonnées à partir des anciennes, par application des équations de la transformation,

(ii) l'interpolation des valeurs obtenues en fonction d'un voisinage variable, car la grille des coordonnées réelles ne correspond pas à celle des coordonnées entières de l'image de départ.

Cette interpolation est une convolution dont le noyau est appelé fonction d'interpolation. Bien que la fonction idéale soit la fonction sinus cardinal: *(sinx)/x*, le nombre infini d'opérations effectuées pour la calculer, explique l'utilisation de fonctions d'interpolation plus simples: plus proche voisin, bilinéaire, ou bicubique. [MOUG86] a d'ailleurs retenu la fonction bilinéaire pour leurs applications, ce qui dénote au passage le niveau de précision requis.

La méthode de calcul des coordonnées des points tournés est donc particulière. Partant d'un point de la grille de l'image tournée, elle consiste à calculer la position dans l'image originale de ce point. Ce calcul est effectué par rotation inverse:



A partir du point de coordonnées entières *(x,y)* de l'image finale, on obtient par rotation inverse le

point de coordonnées réelles *(x',y')* appartenant à l'image initiale:

$$\begin{bmatrix} x' - x_0 \\ y' - y_0 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \times \begin{bmatrix} x - x_0 \\ y - y_0 \end{bmatrix}$$

Les coordonnées réelles successives se calculent par la méthode incrémentale que voici:

$$\begin{aligned} x'[x,y] &= x'[x-1,y] + \cos\theta \\ x'[x,y] &= x'[x,y-1] + \sin\theta \\ y'[x,y] &= y'[x-1,y] - \sin\theta \\ y'[x,y] &= y'[x,y-1] + \cos\theta \end{aligned}$$

A partir de ces équations, il est possible de calculer les nouvelles coordonnées des pixels en utilisant exclusivement des additionneurs.

Voyons maintenant comment est déterminé le niveau de gris de ce point *(x',y')* dans l'image originale, et qui est donc aussi le niveau de gris du point *(x,y)* de l'image finale. Ce niveau de gris noté *f(x',y')* est calculé par la fonction d'interpolation bilinéaire de Lagrange, comme suit:

$$f(x',y') = \sum_{i=0}^{1} \left(\sum_{j=0}^{1} C_{ij}(x',y') \times f(E(x')+i, E(y')+j) \right)$$

avec $E(x')$ = partie entière de x'

La matrice des coefficients est la suivante:

$$C_{00}(x',y') = (1-\alpha) \times (1-\beta)$$

$$C_{01}(x',y') = (1-\alpha) \times \beta$$

$$C_{10}(x',y') = \alpha \times (1-\beta)$$

$$C_{11}(x',y') = \alpha \times \beta$$

où $\alpha = x' - E(x')$ $\beta = y' - E(y')$

Le niveau de gris du point *(x',y')* est donc calculé par interpolation bilinéaire des niveaux de gris de ses quatre premiers voisins appartenant à la grille entière de l'image originale. La saisie des quatre voisins est facilitée par un découpage *ad hoc* de la mémoire d'image, en quatre bancs séparés. Un générateur des positions de ré-échantillonnage *(x,y)* calcule les valeurs *(x',y')* puis les coordonnées des quatre voisins entiers de ce dernier. Quatre multiplieurs en parallèle, suivis d'un additionneur[1], permettent le calcul de cette interpolation [MOUG86].

Architecture systolique

Dans le modèle systolique, les données sont "pipelinées" à travers les processeurs voisins. Les processus s'effectuent par des instructions locales à chaque processeur. La synchronisation est globale. Ce concept est fréquemment utilisé pour développer des processeurs dédiés à un algorithme précis. Nous avons retenu ce type d'architecture pour un processeur rapide de rotation de la machine PASTIS. Les raisons de ce choix sont les suivantes:

(i) l'*importance du temps de réponse* du processeur de rotation fréquemment utilisé: les modèles SIMD et MIMD souffrent du nombre d'éléments traités au moment de leur chargement et déchargement,

(ii) du fait de l'implantation de plusieurs processeurs dédiés à chaque traitement, le concept MAD, l'aspect pipeline assure un flot continu des données à travers la machine,

(iii) enfin, le modèle systolique est parfaitement adapté à l'algorithme de rotation que nous proposons qui pour un pixel entré génère un nombre important d'opérations extrêmement rapides.

Nous n'avons rien trouvé sur l'existence d'une application du type systolique de la rotation, bien que nos arguments se retrouvent souvent dans la littérature. Notons toutefois que notre approche ne devient possible qu'une fois accepté le biais d'une imprécision de la rotation obtenue.

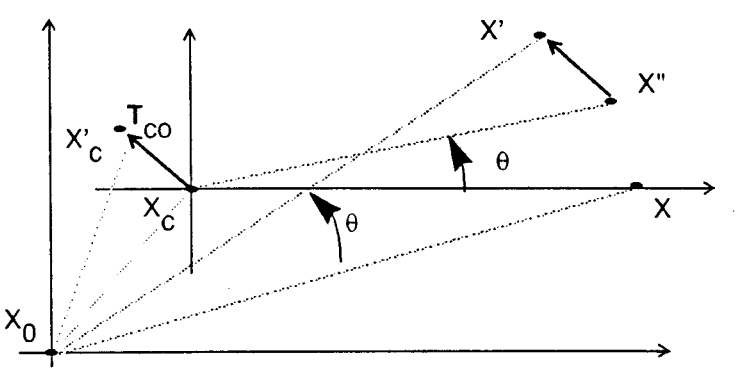
2.2 Parallélisation du processus de rotation

2.2.1 Décomposition d'une rotation

Comme nous l'avons vu au chapitre précédent, une rotation autour d'un centre X_0 peut se décomposer en:

(i) une rotation autour d'un centre X_c,

(ii) et une translation relative à la rotation de X_c autour de X_0.



Cette décomposition du processus de rotation trouve son application dans un processeur systolique.

1. La projection systolique de ce modèle éviterait le recours à un additionneur quatre entrées que préconise [MOUG86]. De même, cette projection permettrait probablement un gain sur le débit du circuit...

Les lignes de pixels de l'image entrent à la file dans ce processeur. On effectue sur chacune une rotation de centre: le centre de la ligne. On translate alors la ligne tournée selon un vecteur dépendant du numéro de la ligne entrée. L'aspect pipeline du processeur systolique nous assure une bonne cadence de rotation, une fois le processeur chargé.

2.2.2 Projection systolique de cet algorithme

Brièvement, nous décrivons les opérations effectuées dans le processeur: la rotation puis la translation.

Première phase: la rotation

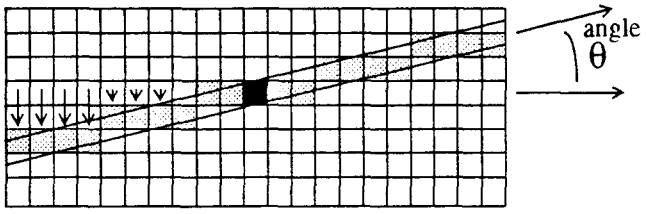


figure 3 Rotation effectuée pour chaque ligne entrante

Cette figure décrit la première phase de l'opérateur, qui effectue pour chaque ligne une rotation d'angle θ autour du centre de celle-ci. Cette rotation se fait à l'aide d'une mémoire qui contient une ligne *pré*-tournée de l'angle voulu. Le temps pour effectuer cette rotation s'exprime donc en temps d'accès aux N.sin(α) lignes contenues dans cette RAM, où N est la longueur de la ligne tournée et α l'angle.

Deuxième phase: la translation

Dans le processeur systolique de rotation, les lignes de l'image d'entrée arrivent les unes derrière les autres. D'après les équations précédentes, la translation effectuée pour tous les pixels d'une même ligne dépend de l'occurrence de celle-ci. La translation doit être vue comme un *itinéraire emprunté par les lignes de pixels*. Cet itinéraire est composé uniquement de décalage de haut en bas et de gauche à droite dans une fenêtre de l'image de sortie.

Le nombre absolu des décalages reste borné par les valeurs N.cos(α) en largeur et N.sin(α) en hauteur. Par contre, pour deux lignes entrées successives de centre X_c et $X_{c'}$, on a:

$$\Delta T = T_{c'o} - T_{co} = R \times \left(X_c + \begin{bmatrix} 0 \\ lig \end{bmatrix} - X_0 \right) - R \times (X_c - X_0)$$

$$\Delta T = R \times \begin{bmatrix} 0 \\ lig \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \times \begin{bmatrix} 0 \\ lig \end{bmatrix} = \begin{bmatrix} -lig \times \sin\alpha \\ lig \times \cos\alpha \end{bmatrix}$$

ce qui signifie que le nombre de translations effectuées dans le *BitMap buffer* pour repositionner les lignes entre elles, est limité par sin(α) et cos(α) pour deux lignes successives (lig=1).

Le parallélisme temporel apparaît dans ces deux phases qui s'enchaînent sans se recouvrir. Quant au parallélisme spatial sur les pixels, il est double: les pixels d'une même colonne partagent les mêmes données à des instants successifs dans la phase de rotation, tandis que tous les pixels d'une même ligne sont translatés ensemble dans la phase de translation.

2.2.3 critique sur la qualité des images

Dans toute rotation effectuée sur un écran, on observe une disparition du nombre de points. Celle-ci est due au ré-échantillonnage (interpolation) en pixels d'une droite horizontale tournée d'un angle donné qui crée donc un recouvrement de certains pixels dans l'image tournée.

Dans notre modèle, la fonction d'interpolation au plus proche voisin est appliquée deux fois, générant ainsi une double erreur de ré-échantillonnage: les valeurs réelles, obtenues de la multiplication par des fonctions trigonométriques, sont converties une première fois, lors de la rotation autour du centre de la ligne, puis une deuxième, lors de la translation.

Nous avons déjà étudié ce phénomène, et comparé les résultats de notre processus de rotation par rapport à ceux d'une rotation précise. L'épithète précise sous-entend rotation et interpolation au plus proche voisin, bien que cette interpolation soit loin d'être la meilleure. Nous conseillons pour les lecteurs intéressés par cette étape d'évaluation d'un processus approché et de comparaison au processus exacte, de se reporter au document interne [DELE89].

Ces simulations, effectuées sur Station SUN, ont permis de mesurer la qualité des images tournées, obtenues par la partition de la rotation de chaque ligne en une rotation par rapport au centre de la ligne, avec arrondi entier pour connaître la position du nouveau pixel, suivie d'une translation (toujours avec arrondi entier) identique pour tous les pixels de la ligne. Cette séparation garantissait une simulation rigoureuse de notre opérateur. Nous avions utilisé des images de taille NxN, où N=256.

Ces simulations ont permis l'adoption du processus de rotation approchée, puisqu'il n'induit pas une perte importante du contenu "visuel" de l'image: la mesure de ce contenu "visuel" s'est faite sur des critères morphologiques, notamment différence symétrique entre les deux rotations, quantité de points relative de cette différence symétrique,... Néanmoins, il apparaît que ce processus ne donne pas de bons résultats pour les angles approchant $\pi/4$; ce qui se conçoit aisément pour une méthode approchée de rotation.

2.2.4 Itérations d'angle

Afin de remplacer la rotation approchée d'angle $\pi/4$, nous avons imaginé d'itérer une rotation d'angle diviseur de celui-ci: par exemple, $\pi/8$, $\pi/12$ ou $\pi/16$. Les résultats obtenus prouvent que les angles α proches de $\pi/4$ ont intérêt à être obtenus par itération: on itère n fois une rotation d'angle α/n, plutôt que de faire la rotation d'angle α. Cette solution sera retenue puisqu'elle permet d'obtenir plusieurs rotations à partir d'un processeur de rotation $\pi/8$ par exemple. Nous verrons par la suite combien cette possibilité de fixer une borne supérieure de la valeur d'angle peut être importante dans un mécanisme de rotation dédié; nous pourrions dire *câblé*.

Pour clore cette synthèse succincte de nos simulations, il faut préciser que l'inconvénient majeur de notre processus vient de la perte de connexité de certains pixels entre eux: deux pixels reliés, en topologie 8 voisins, peuvent se retrouver "distants", i.e. non voisins. Pour l'optique de l'utilisation par PASTIS 90 de cette rotation, cette perte de connexité importe peu. Pour une utilisation autre (impliquant un suivi de contour, par exemple), elle pourrait se révéler catastrophique. L'utilisation d'opérateurs associatifs, dans toute la chaîne de production d'information de Pastis, assure la cohérence globale entre le type d'information extraite (précise ou approchée) et son exploitation.

2.2.5 *Méthodes analytique versus pixel*

Avant de présenter l'architecture que nous avons conçue, il nous semble important de distinguer essentiellement deux méthodes pour appliquer ce processus de rotation. La première *analytique* consiste à employer des vecteurs et à effectuer des opérations arithmétiques entre vecteurs. Cette méthode est relativement lourde puisqu'elle nécessite un usage intensif d'unités arithmétiques.

Toutefois, la rotation approchée ne requiert plus l'utilisation de multiplieurs, et on montre facilement que celle-ci s'obtient avec des additionneurs uniquement, pour peu qu'on ait stocké les vecteurs correspondant à la translation et ceux de la rotation dans une RAM. Cette méthode se rapproche d'une rotation incrémentale, vue par exemple chez [MOUG86]. Cette rotation incrémentale est d'ailleurs d'un usage fréquent en imagerie.

La description architecturale d'un tel processeur est simple: il possède deux additionneurs, fonctionnant en virgule fixe et pipelinés *(cf. figure 4)*. Ceux-ci calculent la nouvelle adresse dans la mémoire image de sortie des pixels noirs: l'un calcule l'adresse après rotation, l'autre après translation. La décomposition en deux phases, rotation et translation, évite les multiplieurs au temps de réponse plus long. Le calcul globalement effectué peut être distribué sur plusieurs processeurs.

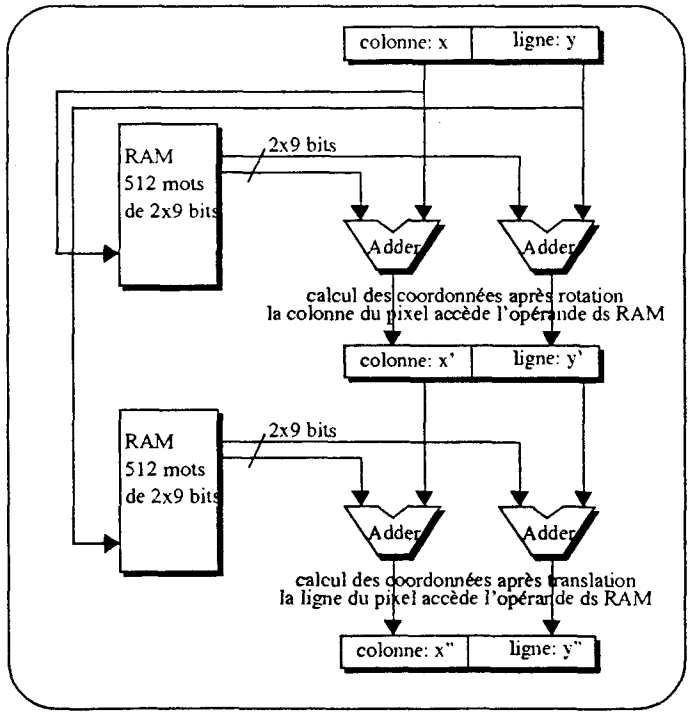


figure 4 MAD de rotation arithmétique

Les coordonnées nécessaires pour la rotation et la translation sont stockées dans deux RAMs séparées. Les coefficients pour la rotation sont les mêmes pour tous les pixels d'une même ligne, ceux de la translation pour tous les pixels d'une même colonne. Aussi ces deux RAMs sont de taille définie: 512x2x9 bits, plus généralement $2.N.log_2N$ bits. Le calcul effectué par les additionneurs l'est en virgule fixe.

un ou plusieurs angles?

Les données contenues dans les 2 RAMs (rotation + translation) sont associées à l'angle de rotation. Changer ces données revient à modifier l'angle, ce qui permet d'obtenir un processeur adapté à plusieurs angles. L'intérêt semble évident pour PASTIS: d'un point de vue coût puisqu'il évite la fonderie de n circuits (si on désire n rotations différentes). Néanmoins, nous avons signalé que l'itération du processus de rotation approchée pour un angle donné n'est pas source d'une perte significative d'information. Aussi la possibilité d'avoir plusieurs angles de rotation n'est pas fondamentale.

Cette solution est d'autant plus envisageable, que le nombre de pixels noirs présents dans un dessin au trait est relativement faible (de l'ordre de 1%). Une telle méthode, qui ne traite que les pixels noirs, ne garantit pas un temps de réponse constant. Comme pour l'architecture proposée par Arabnia, ce temps de réponse est proportionnel au nombre de pixels noirs présents dans l'image d'entrée. Dans le pire cas, pour un écran complètement noir (NxN pixels), il est de NxN le temps de cycle du processeur.

Mécanisme d'accès aux pixels noirs

Un tel processeur a besoin des adresses des pixels noirs présents dans l'image. Ceci peut se réaliser par accès à des mots complets (8 ou 16 bits) sur le bit-map de l'image d'entrée. Le mécanisme délivre les adresses des pixels noirs dans ces mots. La mémoire image d'entrée est alors accédée à une vitesse très importante; on peut penser à un mécanisme du type DMA. Le dessin pipeline simple de ce processeur permet un haut parallélisme lors de l'accès à cette image d'entrée. Des buffers, stockant les adresses des points tournés, éludent momentanément le problème du flot irrégulier et aléatoire des accès à la mémoire image de sortie; leur taille doit pouvoir s'estimer de manière stochastique.

La deuxième solution consiste à travailler non plus avec les coordonnées des pixels, mais directement sur les pixels: il s'agit d'une méthode que nous qualifierons de *pixel*. L'idée est de tourner simultanément tous les pixels d'une même ligne: processus à haut parallélisme. Cette proposition architecturale concrétise la méthode de rotation approchée, et l'idée d'un opérateur de rotation "câblé".

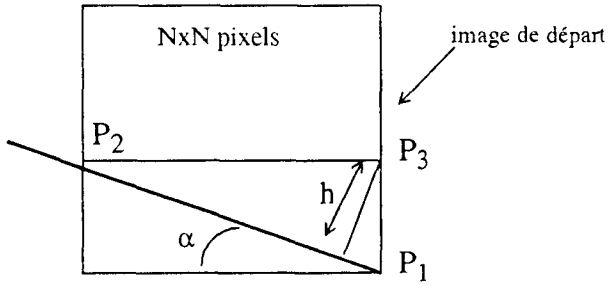
2.3 Architecture du MAD de Rotation Rapide:

Nous présentons ici l'architecture de notre processeur de rotation rapide (PRR) capable de tourner P pixels simultanément, et du MAD de rotation qui regroupe plusieurs PRR. Les objets manipulés par un PRR sont des lignes composées de P bits successifs; les opérations effectuées sur ces objets sont uniquement logiques, et donc rapides comparées aux opérateurs arithmétiques précédents. La technique du pipeline est largement utilisée, pour recouvrir de façon temporelle des actions identiques mais effectuées sur des données différentes.

De plus, l'aspect parallèle n'est pas négligé pour tenir compte de la possibilité de tourner une ligne entière, ou tout du moins une grande partie, simultanément. Ainsi le PRR tourne un bloc de P pixels en un temps constant, relativement faible. Cette partie concerne l'aspect fonctionnel du PRR. Une publication interne regroupe les aspects architecturaux fins, *i.e.* la conception des ASICs [DELE91].

2.3.1 La rétention des pixels

Les MADs étudiés doivent travailler de concert dans la machine PASTIS. Cet aspect a donc contraint le choix d'une architecture basée sur les concepts de pipeline et de parallélisme: c'est-à-dire systolique. Ces MADs sont donc alimentés en données par des blocs de pixels de taille importante. Cependant, dans le cas du processus de rotation, si l'on envisage une rotation ligne par ligne de notre image, nous sommes confrontés à un problème de rétention des pixels. Le schéma suivant montre que la ligne à laquelle appartient le pixel P_1 ne sort qu'une fois entré et tourné le pixel P_2:



Si P_2 est entré, P_3 est donc entré lui aussi, mais doit attendre un certain temps avant de sortir. Ce temps est proportionnel à h, où:

$$\mathbf{h = N \times \sin(\alpha).}$$

Ce temps limite donc l'accès à l'image tournée entre l'initialisation du processus de rotation et la possibilité d'accès au bas de l'image déjà tourné, c'est à dire au segment $[P_1,P_2]$. Le MAD de rotation, ainsi que les PRR, doivent donc posséder un mécanisme interne de stockage temporaire de l'information dont la taille dépend respectivement de $N\sin(\alpha)$ et $P\sin(\alpha)$.

2.3.2 Mise en parallèle des PRR

Un MAD de rotation est une structure d'accueil de plusieurs PRR. L'utilisation d'une mémoire de travail indépendante pour ce MAD est une nécessité qu'explique le partage du bus d'images pendant la

phase de rotation.

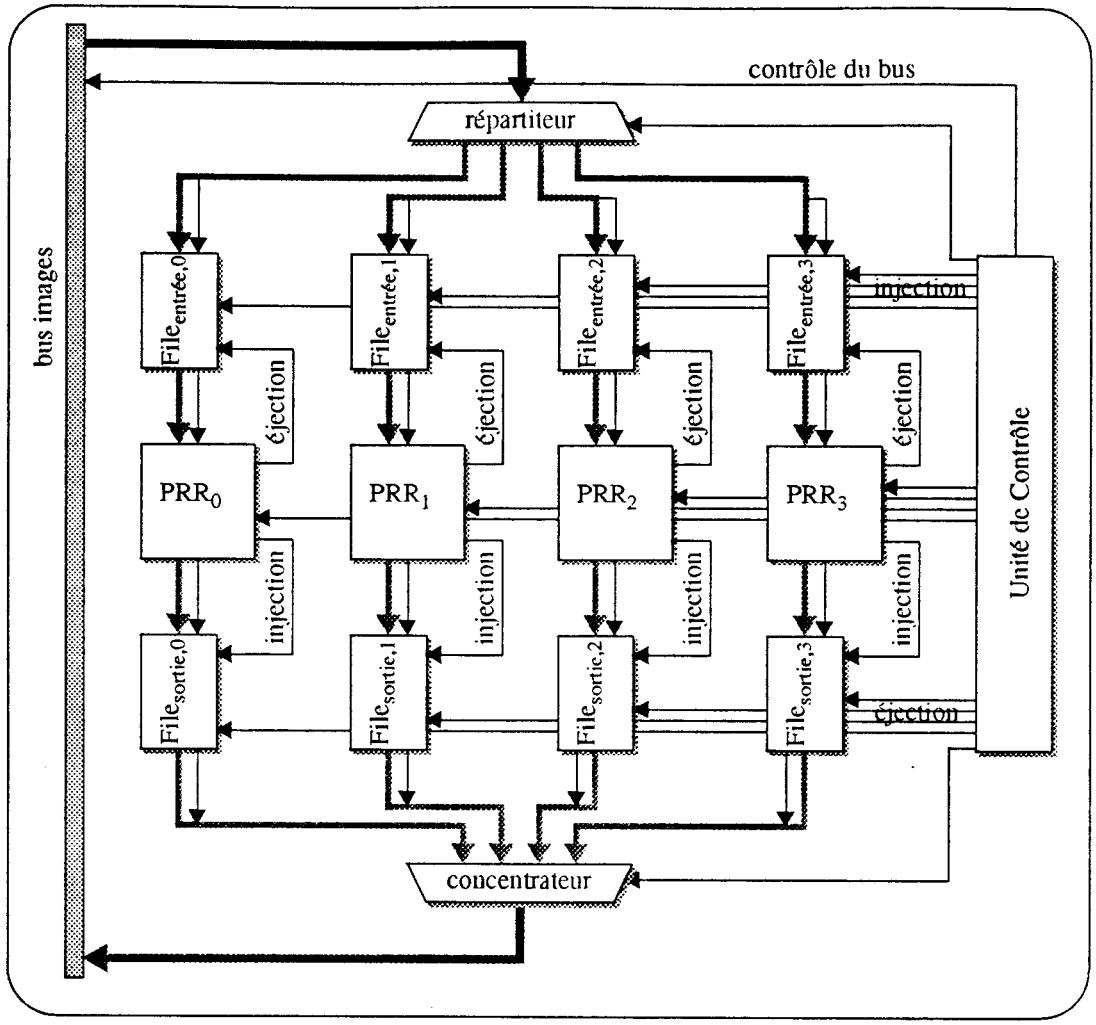


figure 5 Schéma synoptique du MAD de rotation

Pour l'image d'entrée, si celle-ci est de taille N^2 (N=512), il est possible de la décomposer en M blocs de P pixels (avec N=MxP) dans le sens de la largeur. Le traitement est effectivement indépendant, et peut s'effectuer de différentes manières: réalisation de MxN processus de rotation parallèle sur P pixels, ou réalisation par M processeurs de N processus de rotation parallèle sur P pixels. Notre PRR tourne justement N lignes de P pixels.

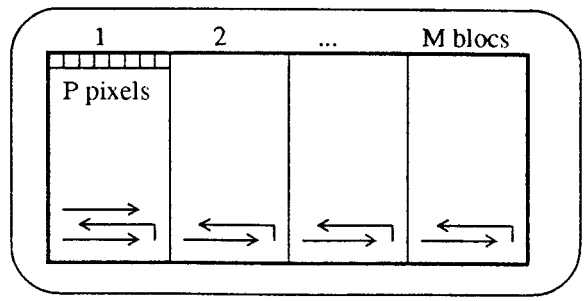


figure 6 Entrelacement de l'image d'entrée sur les bancs

La mise en parallèle de M processeurs permet de diminuer la taille du mécanisme de stockage temporaire, puisqu'il est proportionnel à la largeur du bloc traité, et donc divisé par M. Ce partage se réalise plus facilement grâce à un découpage identique de l'image d'entrée, qui s'obtient par un découpage de la mémoire en bancs distincts entrelacés *(cf. figure 6)*. Cet accès est géré directement au niveau de la mémoire d'images globale de PASTIS que nous présenterons ultérieurement.

Le MAD de rotation possède donc une mémoire locale. Celle-ci est une file de lignes de P pixels stockées avec leur adresse. Chaque PRR accède une ligne de cette file au début de son processus de rotation de la ligne. Comme ce processus est long (en terme de cycles processeur), l'accès peut être recouvert dans le temps.

Quant à l'accès à une image de sortie, les informations ne seront pas directement copiées *via* le bus images, dans la mémoire d'images. La mémoire locale peut être partagée entre les écritures d'un des PRR vers cette mémoire, et les lectures d'un bloc de P pixels par l'un des PRR. Notons que ce partage est possible, puisqu'un certain nombre d'étapes sont réalisées entre l'entrée d'une ligne à tourner et la sortie d'une ligne tournée: le PRR possédant un mécanisme interne de stockage temporaire. Autre solution, l'existence d'une file en entrée et d'une file en sortie par PRR: nous choisissons cette dernière solution.

Pour terminer, la figure 5 présente des mémoires double-port utilisées comme mémoire de travail, files d'entrée et de sortie. Nous retenons cette solution pour recouvrir les opérations d'entrées-sorties des PRR et les lectures-écritures des images, à tourner ou tournées, dans la mémoire d'images de Pastis *via* le bus d'images. Ainsi le partage de l'accès au bus est facilité, sans réduire les performances des PRR gourmands en entrée-sorties.

Si l'accès à une ligne complète (N pixels) est possible à chaque cycle de rotation, les M PRR tournant chacun les P pixels en un temps τ, alors le MAD de rotation est capable de tourner une image de N^2 pixels en $N\times\tau$. Le processus de rotation s'effectue donc en temps constant, indépendant du nombre de pixels noirs. Tous les pixels sont tournés, blancs et noirs. Les PRR sont, nous allons le voir, construits autour d'opérateurs rapides: décaleurs 1 bit, portes logiques, et offrent ainsi une rapidité importante pour exécuter leurs tâches. Les limites imposées le seront donc par les accès aux pixels, tant en entrée qu'en sortie. Ces limites se rencontrent toujours lorsqu'il est fait appel aux techniques systoliques, qui demandent un grand débit d'entrées-sorties.

2.4 Fonctionnement du PRR

Chaque rotation se décompose comme nous l'avons présenté au paragraphe *2.2*: rotation autour de son centre d'une ligne quelconque, et translation de vecteur dépendant de la position de la ligne par rapport à la ligne médiane. On suppose qu'un PRR est "câblé" pour la rotation d'un angle donné (e.g. $\pi/16$); nous examinerons dans l'exposé les difficultés architecturales qu'induirait la conception d'un PRR à plusieurs angles de rotation.

2.4.1 Architecture interne

Le PRR est un processeur à deux étages de pipeline: l'unité de rotation et l'unité de translation, aussi appelée *BitMap buffer*. Celles-ci travaillent de manière synchrone, et avec un recouvrement pipeline

interne de leurs opérations.

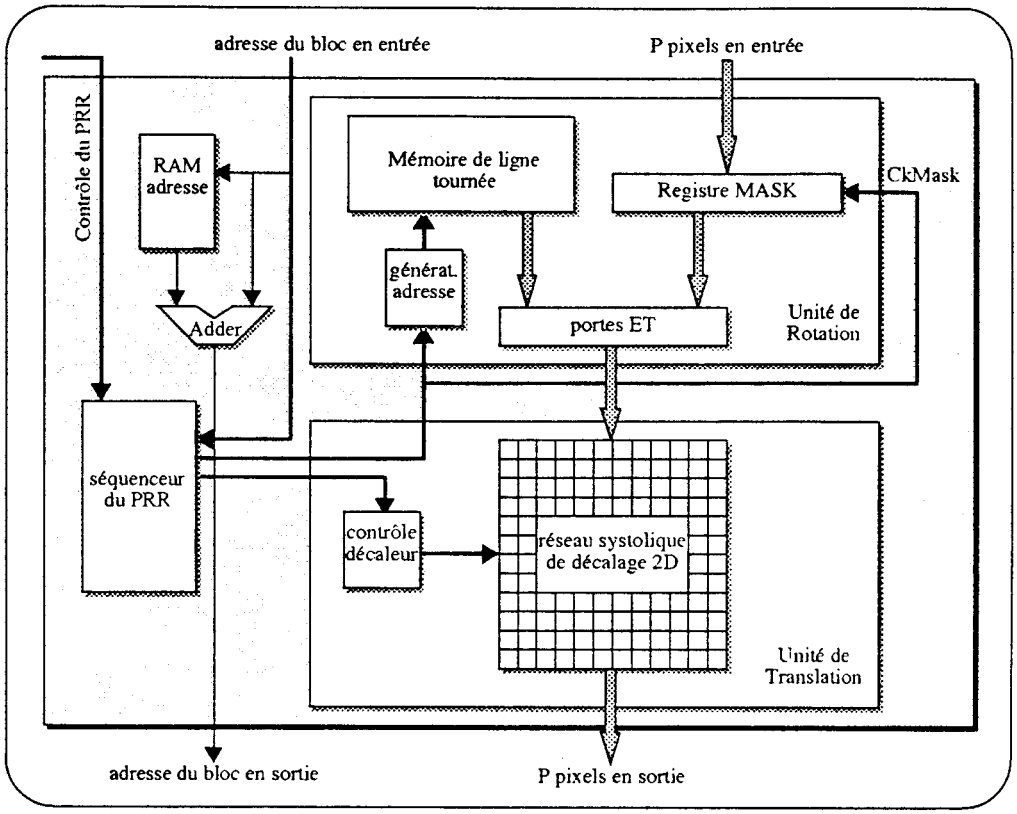


figure 7 Synoptique d'un processeur de rotation rapide

Chaque PRR dispose sur le synoptique de son propre séquenceur qui contrôle l'enchaînement des opérations de chacune des unités; celui-ci peut être le même pour l'ensemble des PRR. De même la génération des adresses des blocs d'entrée, et surtout de sortie, se fait au niveau du PRR; on pourrait l'effectuer au niveau du MAD qui distribuerait les adresses à ses PRR. Le but de cette décentralisation est de démontrer la faisabilité d'un PRR isolé, qui tournerait plusieurs parties distinctes de l'image: un seul PRR tourne M parties de NxP pixels. Nous avons retenu cette méthode pour la phase de test de notre processeur.

Unité de rotation

Soit un bloc de P pixels à tourner, ceux-ci sont stockés dans le registre Mask. L'unité de rotation effectue une rotation sur les P pixels entrés, par rapport au centre de la ligne à laquelle ils appartiennent. Pour cela, on dispose d'une mémoire contenant $Pxsin(\alpha)$ mots de P bits (1 pixel = 1 bit).

Dans cette mémoire, appelée *mémoire de ligne tournée*, nous retrouvons une copie de la figure 3. Les lignes de cette mémoire sont adressées consécutivement; pour chaque ligne accédée, on exécute un ET entre chacun des P pixels obtenus et chacun des P pixels du registre Mask. L'opérateur ET permet donc d'obtenir la rotation de la ligne stockée dans le registre Mask, une fois accédée chacune des $Pxsin(\alpha)$ lignes de la mémoire.

Unité de translation

Cette unité contient le *BitMap buffer*[1], dans lequel se trouvent les pixels noirs tournés et des pixels blancs dans les cases n'ayant reçu aucun pixel tourné. Par un mécanisme *ad hoc* de fenêtrage, ce *BitMap buffer* s'insère exactement dans l'image de sortie, à partir du pixel de référence. La figure 8 décrit le mécanisme de déplacement de cette fenêtre.

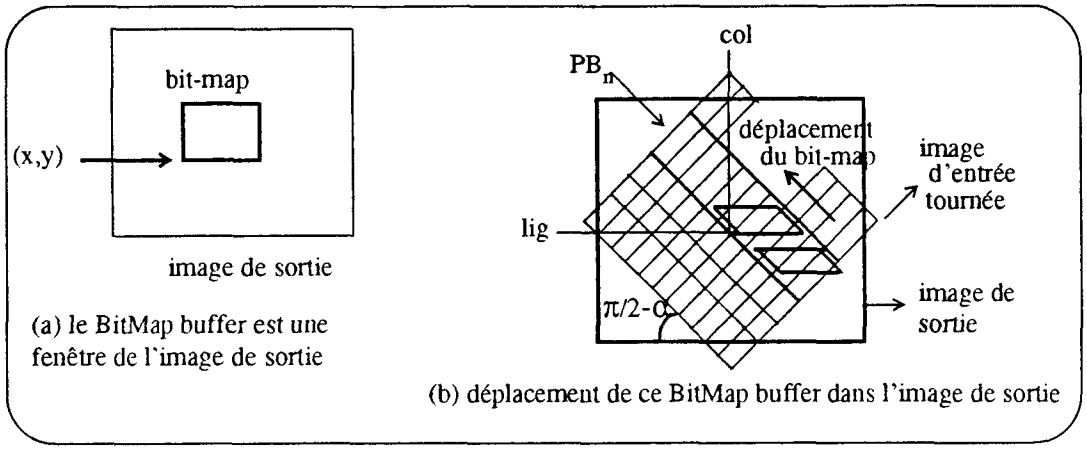


figure 8 Rôle du BitMap buffer dans le processus de rotation

La mise à jour permanente d'un pixel de référence permet de limiter la taille de l'unité de translation. Le déplacement de ce *BitMap buffer* dans l'image de sortie est connu pour un angle donné et la zone de l'image que tourne le PRR. La taille théorique est en largeur de $1+Pxcos(\alpha)$ et en hauteur de $1+Pxsin(\alpha)$, taille confirmée[2] par une simulations fonctionnelle sur laquelle nous reviendrons. Evidemment, si l'angle s'approche de $\pi/2$, cette méthode n'est plus cohérente, en ce sens que le gain attendu est dérisoire.

Ce *BitMap* est donc constitué de P cellules en largeur et de $Pxsin(\alpha)$ cellules en hauteur. Le nombre de cellules par colonne du *BitMap buffer* est donc une limite à l'utilisation du PRR pour des rotations d'angle variable. Si ce buffer possède K cellules par colonne, alors on ne pourra l'utiliser que pour des rotations d'angle α, où $0 \leq \alpha \leq \tau$ avec $1+Pxsin(\tau) \leq K$. La fonction de chacune des cellules est double: translater les nouvelles lignes provenant de l'unité de rotation et sortir les lignes complètes.

Chargement des lignes

L'unité de rotation délivre $Pxsin(\alpha)$ lignes correspondant à la rotation d'une ligne par rapport à son centre. Par le décalage en colonne effectué dans ce *BitMap*, la translation se réduit à un chargement des lignes jusqu'à un certain niveau dans le *BitMap*, niveau précisé par la différence entre le point de référence de ces lignes introduites et le point de référence de l'unité de translation.

Le mécanisme associé est donc un simple transfert de bits (pixels) vers le bas d'une ligne à l'autre; ces transferts sont effectués en mode pipeline, en recouvrement avec l'unité de rotation. Les transferts sont réalisés par des registres à décalage 1 bit, que nous appelons cellules de transfert par la suite.

1. Le terme BitMap fait référence aux données manipulées (1 pixel=1 bit), tandis que le terme Buffer indique l'aspect de stockage temporaire dévolu à cette unité.

2. Le +1 ajouté aux deux grandeurs assure la continuité de celles-ci pour $\alpha=0$ ou $\alpha=\pi/2$.

Sortie d'une ligne

Une fois le décalage effectué, les données stockées dans les cellules de transfert sont chargées dans une cellule mémoire associée à chaque cellule de transfert. On effectue un OU logique entre la donnée provenant de la cellule de transfert et l'ancienne donnée de la cellule mémoire pour ne pas perdre l'information de cette dernière; perte qui serait due au recouvrement de certains pixels après rotation. Dès qu'une ligne peut être extraite du *BitMap* (à la fin de chaque cycle de mémorisation), on décale d'une ligne les cellules mémoire vers le bas; le décalage est effectué *via* les cellules de transfert. La figure 9 décrit le schéma synoptique d'une telle cellule, qui peut communiquer avec ses quatre premiers voisins.

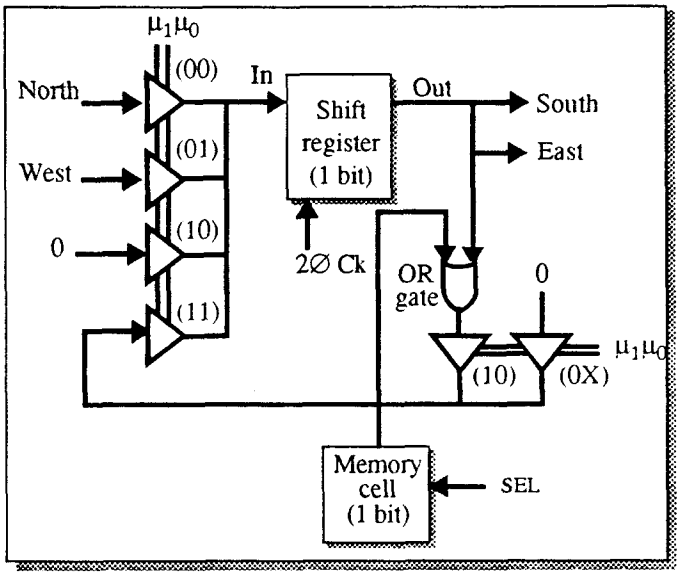


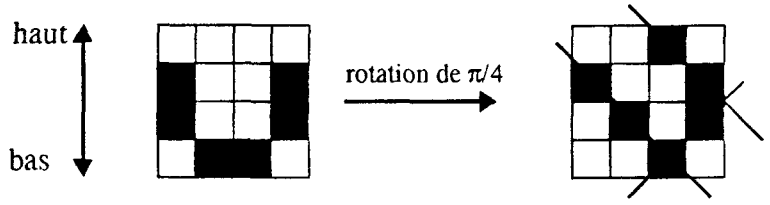
figure 9 Une cellule du BitMap buffer

Mécanisme annexe

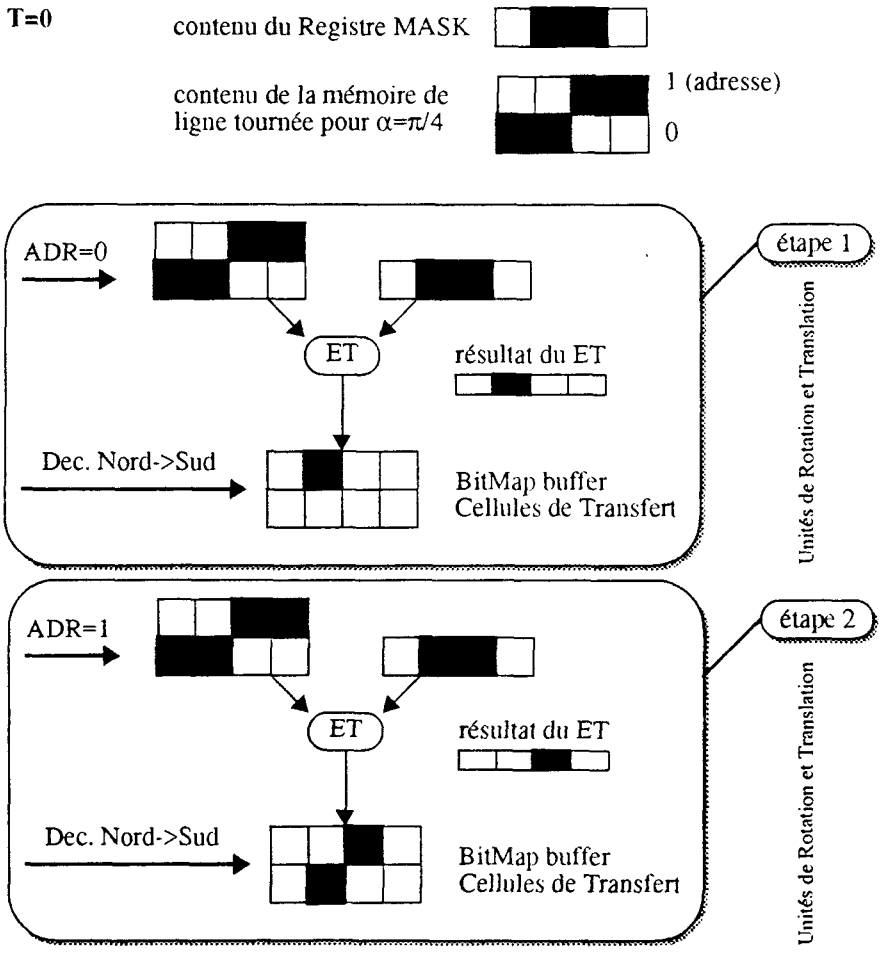
Un additionneur calcule la position réelle du pixel de référence du *BitMap buffer* (pixel le plus à gauche) dans la mémoire image de sortie. Le résultat est nécessaire lors de l'accès à l'image de sortie: pour l'adresse du bloc extrait du *BitMap buffer*, et le décalage à effectuer dans ce BitMap. De plus, les valeurs fournies à ce niveau servent au contrôle des décalages (Nord->Sud, Ouest->Est) lors de la mémorisation des données dans le *BitMap buffer*.

2.4.2 *Exemple de fonctionnement*

Considérons l'exemple suivant, restreint au cas où P=4 pixels. Cet exemple n'est pas du tout précis, et même peut-être faux géométriquement parlant; cependant il donne une bonne idée du processus de rotation et du rôle de chacune des unités. Soit à tourner la figure suivante représentant la lettre *U*:

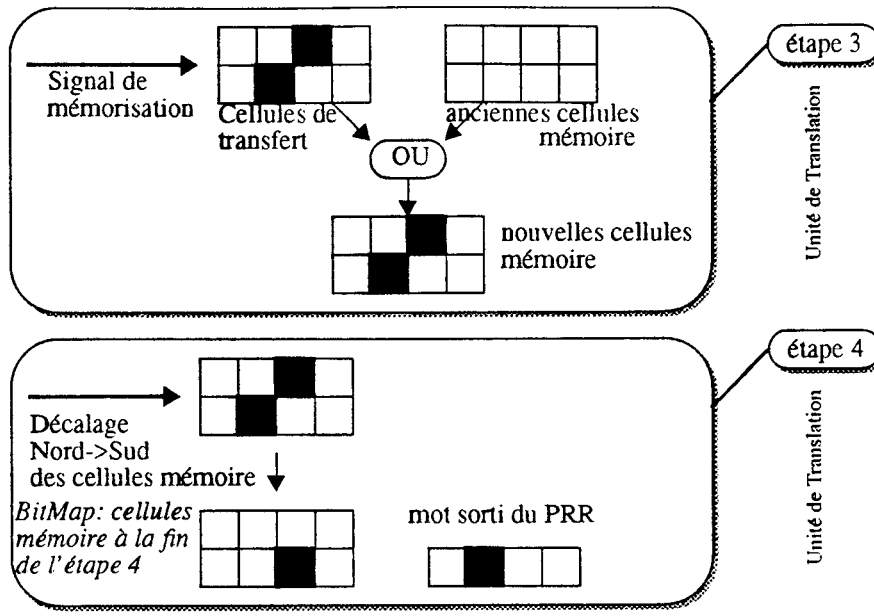


L'angle de rotation est ici de π/4. Nous allons suivre pas à pas le fonctionnement de notre PRR lors de la rotation de cette figure. Nous insisterons particulièrement sur le recouvrement des phases de rotation et de translation entre les deux unités de rotation et de translation. Par contre, la méthode de détermination de l'adresse du bloc de sortie n'est pas décrite; nous donnerons ce calcul par la suite. La première ligne est tournée (ligne du bas). Pour cela, on fait d'abord appel à l'unité de rotation fonctionnant en recouvrement avec l'unité de translation:



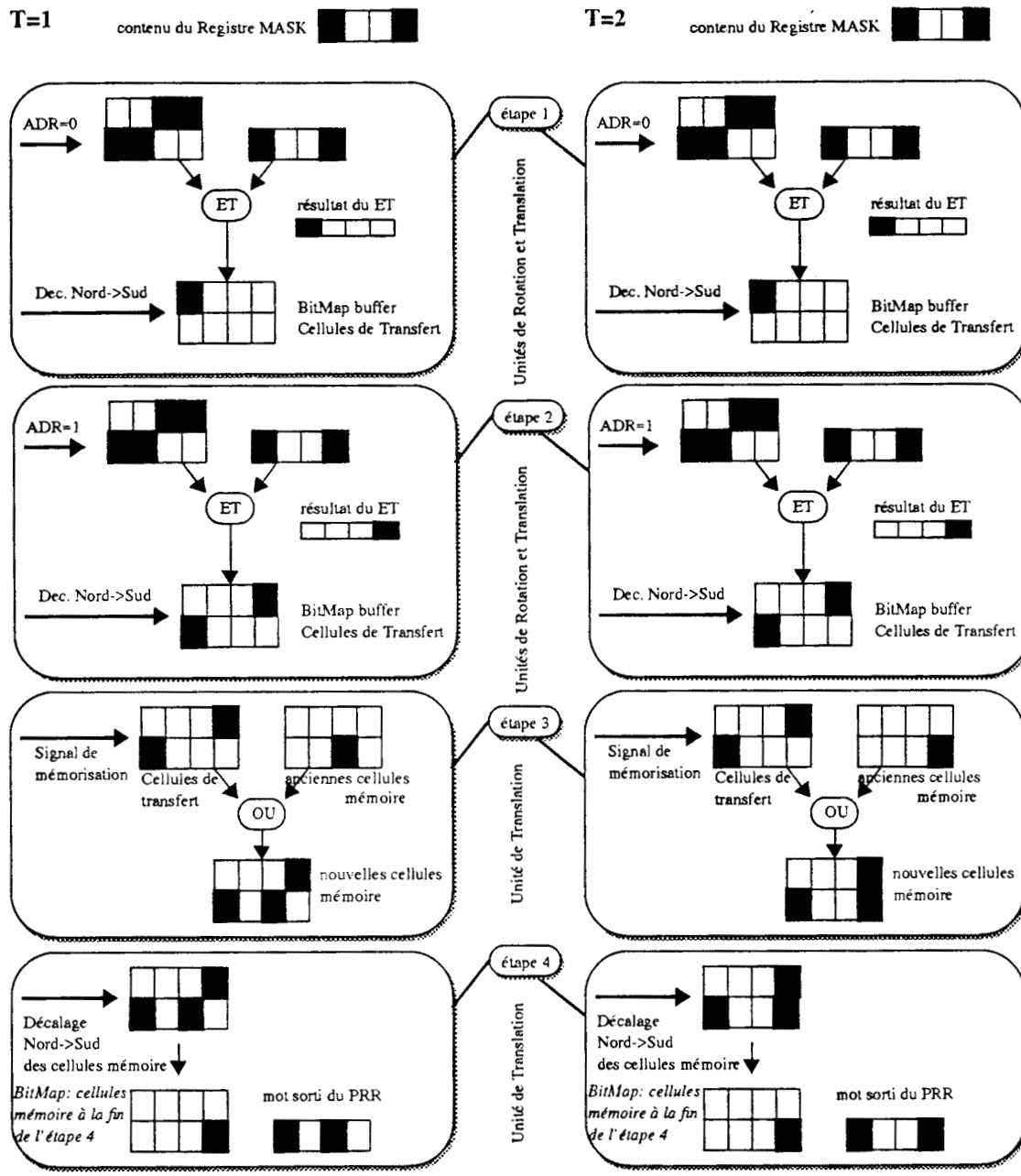
Puis, pour les phases de mémorisation et de sortie d'une ligne, on fait appel à l'unité de translation

uniquement:

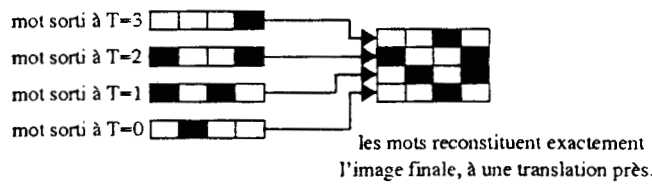


A l'étape 4, on voit donc le mot qui est sorti du PRR (0100) et qui est stocké dans le bas de la mémoire d'image de sortie, tandis que le BitMap buffer garde en mémoire des pixels déjà tournés mais qui sont à extraire dans les étapes 4 des lignes suivantes. Si nous répétons ce processus pour les autres lignes de

notre figure d'entrée:



On voit bien qu'aux étapes suivantes, on n'entrera plus rien dans les cellules mémoire (puisqu'on atteint le haut de la figure d'entrée qui vaut 0000). On peut dès lors reconstituer le dessin à partir des mots extraits aux étapes T=0, 1 et 2, ainsi que du mot à sortir à T=3 (dernière ligne des cellules mémoire du BitMap buffer à T=2). On a donc:

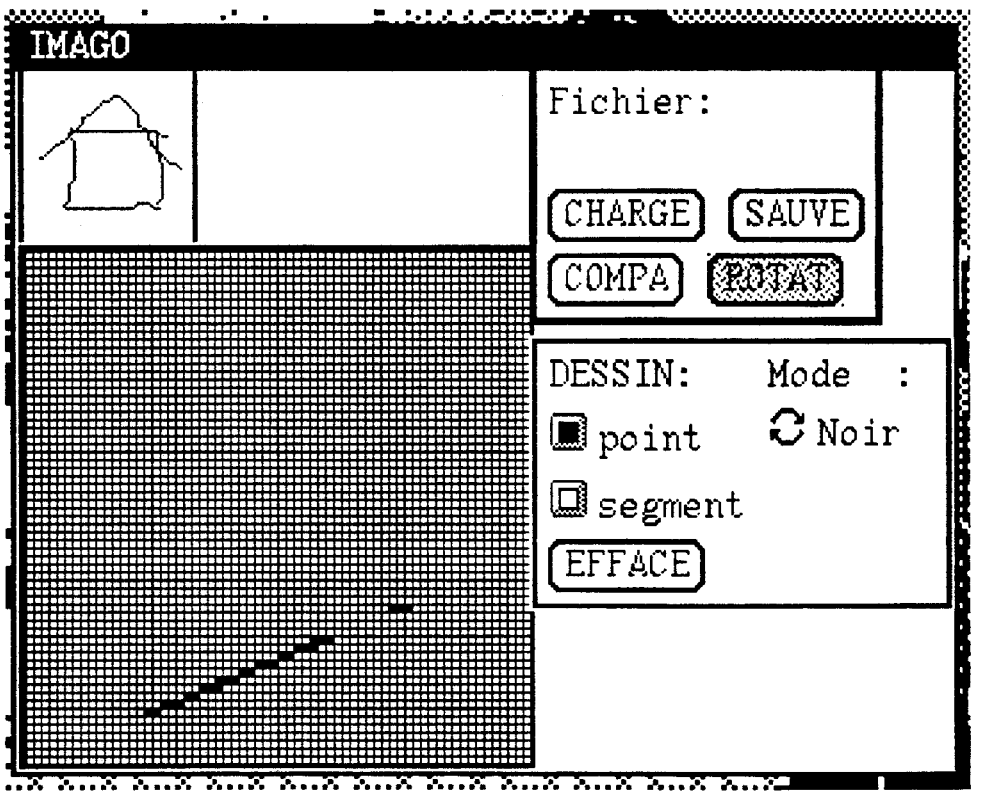


Ceci correspond à la figure attendue à la fin du processus de rotation.

2.4.3 Simulation fonctionnelle

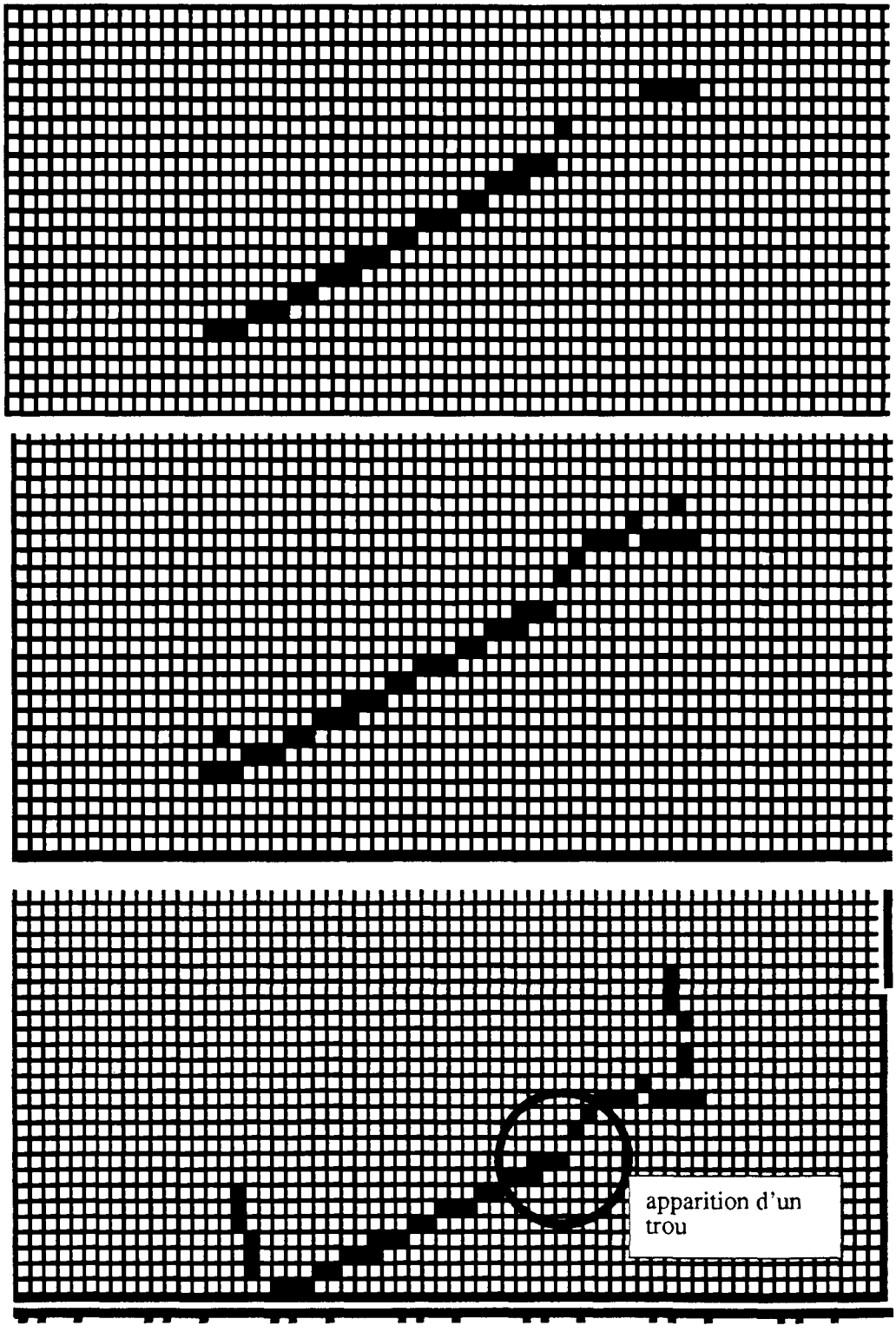
Afin de valider l'architecture de ce processeur, et d'en réaliser les composants de base, nous avons décrit son fonctionnement à l'aide d'un programme C. Cette description aurait pu se faire avec un outil logiciel HDL (*Hardware Descriptive Language*), mais le laboratoire ne possède pas un outil de ce type suffisamment général. De plus, l'enchaînement des séquences s'est avéré suffisamment simple pour qu'une description en langage C se fasse aisément. Nous tenons donc à nous excuser auprès des lecteurs usagers des outils HDL de l'embrouillamini de ce programme, que nous présentons en annexe de cette thèse.

Nous avons ajouté à ce programme une petite interface SUNVIEW, qui nous a permis de suivre et de vérifier *de visu* son bon déroulement. Voici donc, retrouvant une figure qui aura marqué cette thèse, l'évolution du *BitMap buffer* de notre processeur de rotation, au cours d'une rotation de $\pi/8$:



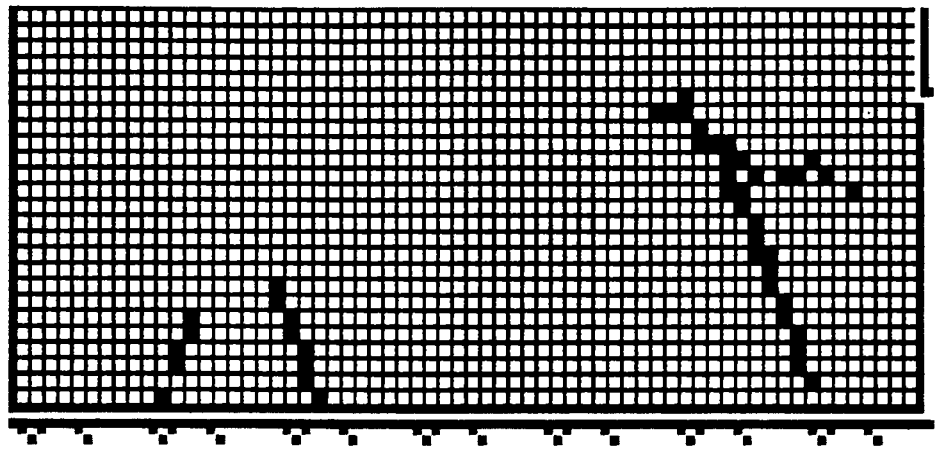
L'image tournée est celle contenue dans le petit cadre en haut à gauche de cette fenêtre (64x64 pixels). Les cellules du dessous contiennent chacune un élément du *BitMap buffer*. Après plusieurs rotation de lignes vides de pixels noirs, la ligne tournée devient celle qui constitue le bas de notre petite maison. On voit donc qu'apparaît dans l'unité de translation, une partie de la droite tournée que contient l'unité de rotation: en effet le bas de la maison est mal dessiné et il manque donc sur son extrémité droite un certain nombre de pixels, qui apparaissent petit à petit au cours des rotations suivantes.

De même qu'apparaissent ces pixels, les cotés de la maison commencent à se dessiner, sur les deux extrémités de cette ligne orientée déjà existante:



Malgrè la qualité des images saisies, on remarquera qu'apparaît un "trou", signe de la disparition possible de l'information de connexité comme nous l'avions envisagée précédemment.

Le dessin suivant donne le contenu du buffer après plusieurs lignes tournées:



On arrive au toit de la maison, et l'on remarque bien que la dernière ligne de ce buffer contient la ligne à extraire au cycle suivant pour laisser la place (par décalage) à une nouvelle ligne orientée bien entendu de $\pi/8$ par rapport à celle qui va sortir.

2.4.4 Recouvrement des accès mémoire en lecture-écriture

Dans le paragraphe *2.4.2*, nous avons vu que la rotation d'un angle α d'un bloc de P pixels requiert Pxsin(α) accès aux lignes de la mémoire de ligne tournée du PRR, auxquels s'ajoutent quatre étapes supplémentaires pour la mémorisation et la sortie d'une ligne tournée dans le BitMap *(cf. figure 10)*.

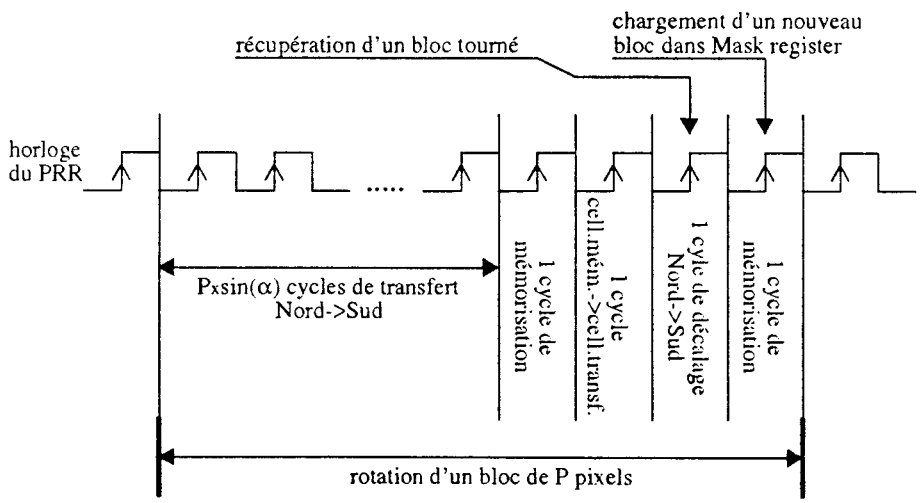


figure 10 Les différentes phases d'une rotation

Pour les valeurs suivantes: P=64, $\alpha=\pi/16$, on a donc Pxsin(α) = 13 étapes, et donc le nombre total d'étapes: 13+4=**17 cycles par rotation d'un bloc de P=64 pixels**.

Grâce aux files séparées d'entrée et de sortie, il est possible de recouvrir les lectures et écritures dans la mémoire de travail. De plus, si on utilise des mémoires double-ports, le chargement des données extérieures (blocs à tourner ou déjà tournés) s'effectue en concurrence avec le chargement du PRR. Ainsi, en utilisant par exemple des données de largeur 32 bits pour le bus Images, la lecture ou l'écriture d'un bloc de 64 pixels prend 2 cycles; la mise en parallèle de M=4 PRR est concevable: 4x2=8 lectures ou écritures. Ceci donne un nombre total de 16 accès sur le bus Images par rotation, nombre inférieur aux 17 cycles/rotation prévus.

Soient M=4, P=64, et **L=32 bits**=largeur du bus. Chaque PRR doit par rotation effectuer une lecture de P pixels, et une écriture de P pixels. Si le bus Images autorise une lecture ou écriture toutes les τ**=100 ns**, alors pour chaque rotation d'une ligne de M.P=256 pixels, le temps requis **T** est de:

$$T = 2 \times M \times \frac{P}{L} \times \tau$$

Cette limite est celle du MAD de rotation à la condition que chaque PRR ait effectivement le temps de faire la rotation en interne. Soit τ' le temps de cycle du PRR, le temps pour effectuer une rotation sur M.P pixels, d'un angle α, avec M PRR en parallèle est donc:

$$T = sup((P \times \sin\alpha + 4) \times \tau', 2 \times M \times \frac{P}{L} \times \tau)$$

Si τ'=30 ns, ce qui peut facilement être atteint, et que $\alpha=\pi/16$, P=64, M=4 et L=32, alors on remarque que $T = sup(510\ ns, 800\ ns) = 800\ ns$. Notons au passage que les accès en entrée-sortie limitent de fait les performances de la rotation rapide. L'évolution des deux valeurs est symptomatique à ce sujet, puisque le temps de calcul (première limite) évolue en $O(P.sin\alpha)$ tandis que le temps limite d'entrée-sortie évolue en $O(N/L)$.

Pour une image de N^2 pixels et avec un MAD de rotation de largeur M.P pixels, c'est à dire M PRR de largeur P pixels, le nombre de cycles de rotations est le suivant:

$$nbre = \frac{N}{M \times P} \times (N + P \times \sin\alpha)$$

Ce nombre provient de la décomposition de l'image N.N, en N/(M.P) blocs de N lignes de M.P pixels. Ces blocs sont tournés successivement. La rotation d'un bloc prend N cycles de rotation parallèle de M.P pixels, suivis de $P.sin\alpha$ cycles de vidage des M BitMap buffer à la fin de ces rotations. L'image est donc tournée en:

$$T_{total} = \frac{N}{M \times P} \times (N + P \times \sin\alpha) \times sup((P \times \sin\alpha + 4) \times \tau', 2 \times M \times \frac{P}{L} \times \tau)$$

Voici un petit tableau donnant une indication des performances prévisibles d'un tel MAD de rotation selon la taille de l'image:

taille de l'image	durée du processus de rotation
N=256	*0,215 ms*
N=512	*0,840 ms*
N=8192	*210,0 ms*

Ce tableau a été calculé pour M=4, P=64, L=32, $\alpha=\pi/16$, τ'=30 ns et τ=100 ns, valeurs qui restent plausibles à l'heure actuelle. Ce qui est intéressant dans ce résultat, c'est d'une part qu'il soutient la comparaison avec les temps annoncés pour diverses machines (de 100 à 20 ms par rotation avec N=256, la prévision est pour notre MAD de 0,2 ms) et d'autre part que ce temps augmente quasi-linéairement[1] avec la taille de l'image.

2.4.5 Développement de circuits ASIC

La conception des deux unités, de translation d'abord et de rotation ensuite, a concrétisé cette étude architecturale. La réalisation de l'unité de translation m'a permis de m'initier au logiciel de CAO SOLO1400. Cela explique certainement le faible niveau d'intégration: 16x8 cellules intégrées par puce. Toutefois, SOLO est un outil *standard cell*, et donc limité en terme d'intégration: chaque composant standard est en effet plus conséquent que ne le serait une cellule dessinée au niveau transistor. On peut raisonnablement envisager une intégration trois, voire quatre fois supérieure en *full custom*: les cellules sont extrêmement simples. De plus, l'utilisation d'un fonctionnement en dynamique du circuit à deux phases d'horloge non recouvrantes éviterait l'usage de bascules statiques encombrantes. L'ordre de grandeur M=4 et P=64 est tout à fait cohérent avec ces remarques.

Ce circuit, conçu au mois de Juin 1990 et reçu fin 1990, reste à tester. L'unité de rotation a été réalisée par deux élèves-ingénieur de l'EUDIL, dans le cadre d'un projet de VLSI. Il va sans dire que j'ai assuré leur encadrement, et que je les remercie à nouveau de leur travail. Cette réalisation s'est concrétisée par l'édition d'une publication interne qui contient trois parties: une partie description du circuit, qui rappelle de très près ce que le lecteur vient de trouver ici; une partie description de l'unité de translation, rédigée par ces deux étudiants et qui couvrent toute la réalisation de leur circuit; une dernière partie qui reprend une parution privée, et concerne la conception et la réalisation de l'unité de translation (appelée pour l'occasion: 2DSA, pour 2D Shift Array) à l'aide de SOLO 1400.

1. Le terme *quasi-linéaire* fait référence au rapport N/MP, puisque N n'est pas *a priori* beaucoup plus grand que MP. De plus ce terme est en dépendance étroite avec les capacités technologiques d'intégration. Notons à ce sujet que le facteur P n'a pas forcément intérêt à être augmenté, du fait de son impact sur le temps mis par chaque rotation de lignes (*cf.* les facteurs à l'intérieur de la fonction SUP pour le calcul de T_{total}).

3 MAD d'extraction de segments

Nous n'avons pas particulièrement étudié ce MAD; ce paragraphe sera donc laconique. Cette étude ne s'imposait pas pour diverses raisons. D'abord l'essentiel du travail consistait à décrire une architecture de processeur de rotation rapide, dû au vide existant dans ce domaine pour ce qui est de la vitesse. Ensuite, les algorithmes d'extraction de segments ont prouvé, largement à notre avis, leur rapidité lors des simulations effectuées sur une station de travail: machine éminemment séquentielle.

3.1 Survol de la complexité des traitements

Dès lors, la réalisation d'une simple carte à base de DSPs aurait accéléré suffisamment ces algorithmes, qui ne posent pas de problèmes de complexité insurmontable. En effet, trois parties se distinguent dans l'algorithme de recherche de droites tel que nous l'avons présenté au chapitre précédent:

3.1.1 Extraction des petits segments

Le parcours des images tournées, avec une décomposition de celles-ci en pavé de taille HxL pixels, suivie du dénombrement des pixels présents dans ce pavé, et, après application d'un seuil, de la sortie d'un pavé contenant ou non un petit segment. L'image ayant été préalablement tournée, ce parcours est donc linéaire sur une bande de H lignes de pixels. Chaque groupe de L colonnes (de H pixels) correspond à un pavé; la suite ne fait intervenir qu'un test global facilement réalisable. Pour H et L variables, l'utilisation d'un incrémenteur est conseillée, mais ne saurait ralentir le fonctionnement qu'en $log_2(log_2(HxL))$ pour le calcul[1] des incréments successifs.

Le nombre de pavé à tester évolue en $O(2xN/HxN/L)$ dans le cas d'une extraction avec décalage (d'où la valeur 2). Comme il y a HxL pixels, le nombre d'opérations est en $O(2.N^2)$, s'agissant d'opérations très simples. La convolution est en $O(N^2.M^2)$ mais avec des multiplicateurs...

Le seul critère susceptible de ralentir le processus tient à la disposition mémoire, que nous abordons justement au paragraphe suivant. Il importe en effet que notre MAD puisse accéder plusieurs pixels d'une même ligne simultanément, quitte à en stocker provisoirement quelques uns dans un long (>>L) registre à décalage de H lignes, par exemple. Ceci n'est pas irréalisable, au contraire.

3.1.2 Génération de pics

Nous avons désigné sous le terme "pic" l'accumulation du nombre de petits segments appartenant à l'une des N/H lignes de l'image décomposée en petits segments. Cette accumulation s'effectue donc en $O(N/HxN/L)$ lectures de l'état 1 ou 0 du pavé considéré. Ici encore, seul un incrémenteur est rendu nécessaire. Ce gain apparaît considérable quand on compare cette méthode à la TH qui fait intervenir des multiplications: $\rho=x.\sin\theta+y.\cos\theta$, pour chacun des N^2 pixels...

On obtient donc une méthode en $O(N^2/HL)$ incrémentations, qui n'est donc pas un problème de calcul mais plus d'entrée-sorties. Remarquons que dans les algorithmes mis en oeuvre pour l'extraction de

1. L'additionneur le plus rapide, comprenez l'incrémenteur, est en log_2N; il s'agit de l'additionneur à propagation de retenues. Pour un pavé de nombre maximum H.L pixels, le codage se fait sur N=H.L, d'où le résultat.

segments le nombre d'entrées, lectures des pixels, est de plus en plus supérieur au nombre de sorties réalisées: extraction d'un petit segment et donc écriture d'état 0 ou 1 d'un pavé (gain en HxL: on lit N^2 pixels, on a N^2/HL pavés), puis génération et sortie du pic (gain en N/L: on lit N^2/HL pavés, on écrit N/H pics i.e. 1 pic par ligne de pavés). Pour H=3 et L=6, et N=512, qui semble des valeurs moyennes, ces gains seraient respectivement de l'ordre de 18 et 170, ce qui n'est pas négligeable.

3.1.3 Parcours de la table d'accumulation

Contrairement à une TH, cette table est une table unidimensionnelle, dans laquelle on va chercher par un algorithme linéaire les pics, parmi les N/H valeurs de cette table. Là encore, le gain sur la TH est appréciable du fait de la dimension de la table qui réduit du même coup la complexité du problème. Pour les n valeurs d'angles traitées, chaque table pourra être parcourue séparément: l'aspect *pipeline* de cette chaîne de traitements prouve une dernière fois la rapidité de l'extraction de droites telle que nous l'avons conçue.

3.1.4 Suivi de petits segments

Une fois détectée l'existence d'une droite, par son pic, la recherche de cette droite est un simple algorithme en deux phases: tassement par projection des petits segments se trouvant à l'intérieur d'une certaine bande horizontale de pavés, puis suivi du premier segment au dernier non isolé pour connaître les coordonnées des extrémités de la droite.

Une fois obtenues les extrémités, l'algorithme est capable de générer les paramètres de la droite de manière très simple. Pour cela, une unité de calcul est indispensable pour évaluer en fonction des données *(α, ligne, col_début, col_fin)* les paramètres *(a, b)* ou *(r, q)* de la droite. Le nombre de droites détectées ne peut pas être connu d'avance, toutefois il est raisonnable de penser qu'il sera faible et qu'une unité de calcul d'un processeur quelconque suffira.

3.2 Architecture utilisée

A la lecture de ces lignes, cette phase d'extraction de segments et de reconstruction de droites apparaît d'une complexité de calcul faible. Un processeur de type DSP suffirait amplement aux besoins en puissance de calcul de cette phase (ex. i860 à 40 Mhz). Une autre solution consisterait à court terme à ne développer que la partie rotation, et à exécuter les calculs de recherche de droites sur la station hôte de ce module. Lors des simulations effectuées sur station SUN, il est apparu que la phase de rotation était de loin la plus longue, bien que cette remarque n'ait rien de scientifique mais soit basée sur une corrélation floue et subjective des observations du programmeur déroulant son algorithme...

Par contre, le point sensible de ce MAD concerne les accès mémoire, dont on peut dire qu'ils vont limiter l'efficacité de la machine. En effet, sachant que HL << N, les deux premières phases auront un nombre d'entrée-sorties sensiblement proches de $O(N^2)$; ce qui pour N=512 pixels reste important. Toutefois, une approche parallèle ou pipeline de ces phases n'est pas impensable du fait du caractère linéaire du traitement, lorsque l'image est vue comme une suite 1D de pavés. Aussi laissons-nous à d'autres le soin d'investiguer les architectures possibles, qui, appliquées à la conception de ce MAD, aboutiraient à la réalisation d'un opérateur de vectorisation extrêmement efficace en terme de complexités temporelle et spatiale.

4 La mémoire d'Images

L'exécution distribué de plusieurs processus implique une étude fine de la mémoire d'images globale. Celle-ci contient à la fois l'image d'entrée et l'image de sortie, pour différents processus gourmands en entrée-sorties. Nous avons décrit les besoins du MAD de rotation, mais venons aussi de montrer que l'efficacité du MAD d'extraction de segments serait limitée par les capacités d'échanges Processeur-Mémoire sur la machine. L'enjeu est d'éviter un problème bien connu en architecture des machines, désigné sous le nom de goulet d'étranglement des machines Von Neuman.

Ce problème provient de la vitesse importante des unités d'exécution par rapport aux unités de stockage. Néanmoins des solutions sont apparues, qui se retrouvent actuellement dans toutes les machines performantes, et donc chères, du marché. La plus employée est celle de l'utilisation d'une mémoire cache locale à chaque processeur. Une solution plus sophistiquée consiste à structurer la mémoire globale, à l'instar des machines vectorielles actuelles.

4.1 Digressions

4.1.1 La mémoire

Le moyen de stockage d'une mémoire dynamique est la capacité d'un transistor MOS; ceci entraîne une perte de cette information due aux courants de fuite de ce transistor (l'information stockée est de l'ordre d'un million d'électrons, ce qui est peu). Il faut donc toutes les micro-secondes aller relire l'information (1 ou 0) et la réécrire pour recharger les transistors. L'avantage d'un tel mécanisme est la place occupée par une cellule mémoire: 3 transistors, ou mieux 1 transistor et 1 capacité. A l'opposé, dans les mémoires statiques, l'information est stockée dans une bascule: il n'y a pas de perte de cette information (sauf en cas de coupure de l'alimentation). L'inconvénient majeur tient à la place occupée par une cellule mémoire: une bascule, i.e. au moins 6 transistors.

Ainsi, du fait des temps de rafraîchissement, une mémoire dynamique est une mémoire lente et qui interdit tout accès lors des rafraîchissements. Ces mémoires extrêmement denses sont donc utilisées comme constituant des mémoires centrales: de taille importante (de quelques mégaoctets à quelques giga...), et de coût relativement faible. Par contre, elles ne peuvent faire face aux demandes des unités d'exécution, puisque trop lentes (rapport de l'ordre de la dizaine dans les temps de cycle, auquel s'ajoutent les temps de rafraîchissement).

Les mémoires statiques permettent à l'opposé de travailler beaucoup plus rapidement; elles sont conçues comme un ensemble de registres. Hélas, chaque bascule occupe une surface plus importante et surtout nécessite un accès à l'alimentation. La densité d'intégration est dès lors sérieusement diminuée. Cependant, on arrive à des temps d'accès à l'information plus compatible (rapport de 2 à 3 pour un opérateur logique à 10ns).

Nous n'abordons pas le cas des machines vectorielles, où les techniques mises en oeuvre (ECL en LSI, temps d'accès de l'ordre de la nano-seconde) semblent infiniment éloignées de notre "pauvre" préoccupation. Cependant, remarquons que les recherches effectuées sur ces machines ont profondément inspiré l'évolution que nous allons retracer.

4.1.2 Evolutions

Les architectes ont donc créé une hiérarchie de mémoires, basée sur la rapidité et le coût de celles-ci. Les structures magnétiques servent au stockage important, la mémoire dynamique au stockage de l'information à accès rapide par rapport aux supports précédents, mais lente par rapport aux unités fonctionnelles qui utilisent des mémoires statiques. La taille des capacités de mémorisation décroît de manière inversement proportionnelle à la rapidité des accès à l'information. La difficulté est de se donner une borne minimale quant à la taille de mémoire requise pour une unité fonctionnelle. Rien ne sert d'utiliser une technique de stockage rapide si, à cause de sa faible capacité, l'unité fonctionnelle ne peut travailler correctement.

La mémoire cache fut créée en considérant ce paramètre important d'efficacité. En effet, dans le cas de programmes usuels, le comportement d'un pointeur de programme (compteur ordinal) est caractérisé par le *principe de localité*. Pour une architecture Von Neuman, le compteur ordinal est le plus souvent incrémenté de 1 à chaque cycle d'exécution, exception faite des branchements, répétitions et interruptions. Ceci signifie que l'action $A_n(t+1)$[1] entreprise à l'instant $t+1$ fait suite à $A_k(t)$ où $k=n-1$ dans la plupart des cas; ceci provient de l'aspect séquentiel des algorithmes tournant sur machines à architecture Von Neuman.

De cette observation est née la mémoire cache de codes, qui contient un certain nombre d'instructions effectuées ou à effectuer -anticipation-, toutes situées autour de l'instruction en cours. Les branchements constituent une épine à cette méthode. Inconditionnels, on peut prévoir que l'instruction appelée le sera effectivement, et donc récupérer celle-ci et quelques unes de ses suivantes. Conditionnels, on ne peut qu'anticiper et récupérer les deux zones possibles de codes à venir (la difficulté est plus grande au niveau du séquenceur lui-même si la machine est pipelinée: on trouve différentes techniques telles que le branchement retardé, le ré-ordonnancement du code,...). Quant aux interruptions, elles ne sont évidemment pas prévisibles et entraînent donc une diminution de l'efficacité.

4.1.3 L'organisation des accès aux données

Ce principe de localité facilite donc le travail des architectes, et prend la forme d'une manne pour les applications dédiées qui répètent souvent les mêmes tâches. En vision, et plus précisément en morphologie mathématique pour prendre un exemple, le code d'un programme réalisant l'union de deux images est très petit. Néanmoins les données sont très larges, de l'ordre du million de pixels.

Profitant de la dualité entre codes et données, le principe de localité s'applique aussi aux données: par exemple dans un tableau, l'accès à l'élément i de ce tableau entraîne bien souvent l'accès à l'élément $i+1$ dans les opérations qui suivent. Il faut remarquer qu'il s'agit du cas général qui s'entend si d'une part le programmeur est un tant soit peu efficace, et possède d'autre part quelques connaissances solides sur le code généré par son compilateur favori. L'une des difficultés majeures rencontrées par les concepteurs de machines vectorielles provient justement des cas où les algorithmes ne font pas des accès symétriques aux données.

Le cas typique en vectoriel est l'accès du type *gather-scatter*, qui pose maints problèmes aux machines vectoriels pipelines. Dans le domaine qui nous intéresse, on peut penser que des accès aléatoires de ce type n'arrivent que très rarement. Or, comme nous l'avons déjà fait remarquer, notre processus de rotation présente justement cette particularité de transformer un accès de type ligne en un accès *pseudo*-aléatoire dû à l'interpolation discrète d'une ligne tournée. Cette transformation empêche

1. $A_n(t)$: n référence la $n^{ième}$ ligne du programme.

l'utilisation d'une structure mémoire classique. Voyons cela en détail.

4.2 Définition de notre mémoire de travail

Tout concepteur de machine s'est posé cette question: comment éviter de pénaliser une unité d'exécution plus rapide que sa source de données? La réponse est encore plus cruciale dans le cas de processeurs systoliques, qui rythment leur marche sur le flot des données. L'étude consiste à profiler au mieux les échanges entre la mémoire et les MADs, qu'ils soient de rotation ou d'extraction de segments, pour en tirer toujours la plus grande force, c'est à dire un transfert maximum de pixels.

4.2.1 Contraintes générales

Nous évaluons le nombre d'images dont aura besoin le MAD de rotation pour travailler efficacement, à environ quatre images 512x512 pixels, voire 1024x1024 par la suite. Puisqu'un pixel est codé sur un bit, la taille de la mémoire requise est d'environ 128K octets (respectivement 512K octets), valeur qui permet d'utiliser des RAM statiques rapides et donc de tenir des temps de cycle relativement faibles. Cependant, comme le MAD de rotation ne sera pas seul, on peut être tenté d'augmenter de manière importante ce nombre d'images: une vingtaine d'images, ce qui ferait 640K octets... La simulation de la machine a démontré qu'avec cinq tableaux image l'ensemble des processus s'enchaînaient correctement, l'aspect pipeline des traitements y étant pour beaucoup. Seule ombre (au tableau, dirions-nous...), parmi ces cinq tableaux, un tableau stocke les segments orientés. Celui-ci prend plus de place, mais nous avons vu qu'il se trouve situé dans le MAD d'extraction de segments.

Une autre contrainte importante consiste à trouver une solution matériel à la réalisation de rotations d'angle multiple de $\pi/2$. Ces rotations correspondent en effet à des lectures en ligne ou en colonne, inversées ou non. Il est donc inutile de les réaliser à l'aide d'un opérateur spécial si notre mémoire de travail le permet. Cette étude fait référence au tableau décalé, introduit lors du projet *Illiac IV (cf. paragraphe 2.1.2* du chapitre *1)*. Nous allons voir comment ces structures rejoignent celles des mémoires caches, dont les contraintes sont proches.

Pour clore les contraintes, nous remarquons que la mémoire d'images devra être équipée d'un opérateur optionnel OU lors des écritures dans celle-ci. Cette contrainte provient du découpage possible de la rotation d'une image en plusieurs rotations de bandes distinctes de cette image.

4.2.2 Structure de la mémoire

La question intéressante tient à l'arrangement de cette mémoire. Chaque PRR travaille par bloc de P pixels, où P aura la valeur[1] 32 ou 64. On peut dès lors penser à utiliser des RAM adressant des octets (mots de 8 bits), et en placer 4 ou 8 en parallèle. Pour les conceptions à venir, il est évident que travailler sur 32 bits serait un bon compromis: taille du mécanisme d'accès au bus par les files d'entrée-sortie dans le PRR, nombre de pattes sur chaque PRR.

1. pour des raisons de simplicité (liée au choix du *standard cells* dans la conception d'ASIC), la valeur de P est 16 pour les prototypes développés. Elle n'implique pas une remise en cause fondamentale du MAD, ni des *timings* envisagés.

Dans cette configuration, l'image est alors entrelacée entre 4 bancs mémoire, comme la figure 11 le décrit avec l'hypothèse[1] suivante: q=8, L=4. Bien que ce modèle soit efficace, puisqu'en un accès mémoire simultanée sur 4 bancs mémoire, le bus (et donc le PRR) récupère 4 mots de 8 bits, i.e. 32 pixels, il ne peut prendre en compte que les accès horizontaux si l'on utilise la disposition mémoire proposée à la figure 12.

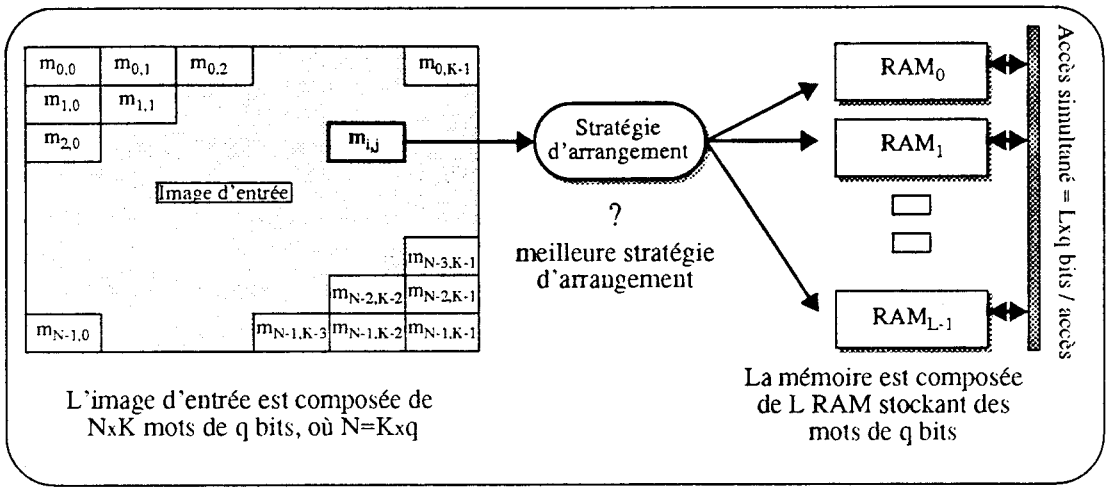


figure 11 Position du problème dans la disposition mémoire

Dans le cas d'un accès à tous les pixels de la première colonne (rotation de $\pi/2$), il faut faire N lectures de mots de q=8 bits sur la RAM_0 pour obtenir nos N pixels: $m_{i,0}$ où $0 \leq i \leq N$.

Pour pallier cet inconvénient, [KUCK68] a présenté la méthode de stockage en tableau décalé utilisé pour la machine *Illiac IV*. Nous avons retrouvé d'ailleurs cette disposition dans les articles très intéressants de [MATI89-1][MATI89-2] sur le fonctionnement et l'architecture d'une mémoire cache, au temps d'accès de 12ns en CMOS (!). L'influence de la mémoire cache est suffisamment significative sur les performances d'une machine générale, pour qu'une telle étude devienne indispensable.

4.2.3 L'arrangement en tableau décalé

L'organisation en tableau décalé permet d'utiliser le plus efficacement possible le bus de *qxL* bits reliant les PRR à la mémoire de travail. Il évite effectivement le cas décrit ci-dessous *(cf. figure 12)* dans lequel la lecture des N pixels de la première colonne demande N lectures de mots de q bits tous stockés dans la même RAM, la RAM_0.

1. Pour la suite, nous considérons évidemment que L et q sont des puissances de 2.

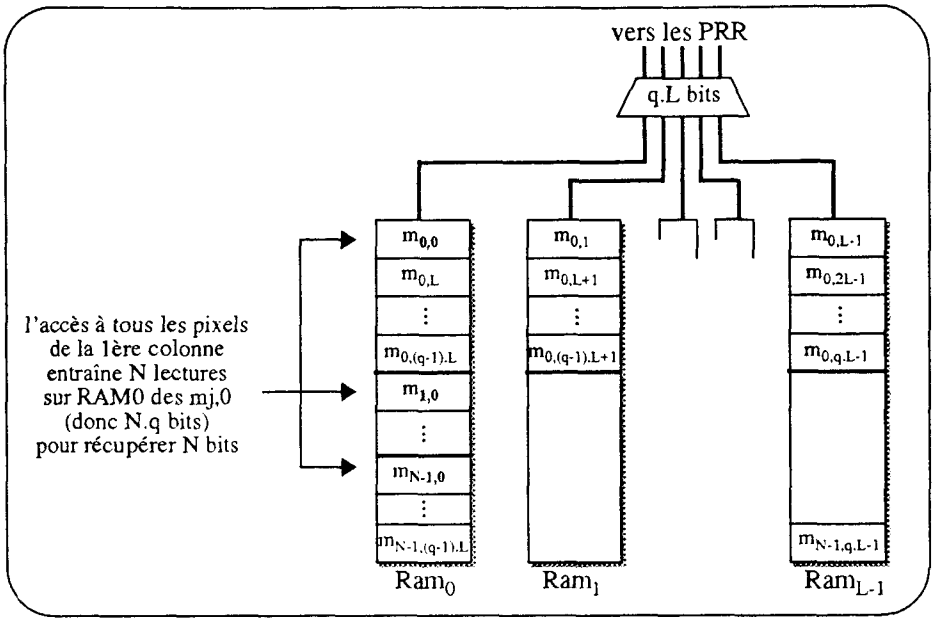


figure 12 Disposition simple inefficace en accès colonne

Cette disposition se retrouve aujourd'hui dans les machines vectorielles. Cette technique est rendue nécessaire par l'importance du *ratio* entre le temps de cycle mémoire et le temps de cycle CPU. Notons que le problème de ces machines vectorielles est semblable au nôtre, à cette différence près que notre machine accède des mots de q bits, où q est petit. Le gain en connexion sera assez important, pour que nous puissions nous permettre d'augmenter le nombre L de bancs mémoire. Au contraire, les machines vectorielles travaillent sur des espaces mémoire étendus et sur des mots de grande taille, au moins 64 bits: situation qui engendre des problèmes de connexion aigus.

quelle mesure d'efficacité?

Si nous voulons mesurer l'efficacité d'une telle méthode, il est nécessaire de former un rapport exprimant la performance d'accès. Soit une mémoire délivrant le plus rapidement (un cycle mémoire τ) toute l'information sans redondance ni information superflue (N pixels seulement), l'efficacité serait ici un rapport indiquant que les N pixels ont été obtenus en un temps τ et sur la base de N données lues (1 donnée=1 pixel=1 bit). Nous prendrons donc un rapport du type suivant:

$$\text{l'efficacité } E = \frac{\text{(Quantité d'information cherchée)x(1 cycle)}}{\text{(Quantité d'information évaluée)x(nbre de cycles de lectures)}}$$

$$E_0 = \frac{N \times \tau}{N \times \tau} = 1 \qquad \text{pour la solution idéale}$$

Pour notre solution idéale, ce rapport égale 1. Pour les moyens de stockage réalisables, ce rapport sera inférieur à 1; sa proximité vis à vis de cette borne supérieure indiquera l'efficacité de notre mémoire. Ainsi, dans le cas d'une disposition de type celle de la figure 12, nos N pixels de la première colonne

sont obtenus en un temps Nxτ (N lectures de RAM_0) et sur la base de Nxq bits lus. D'où l'efficacité d'une telle disposition mémoire:

$$E_1 = \frac{N \times \tau}{(N \times q) \times (N \times \tau)} = \frac{1}{N \times q}$$

On remarque donc que notre efficacité "dégringole" d'un rapport 1/(Nxq), ce qui pour N=512 n'est pas rien.

Dans la machine *Illiac IV*, chaque PE possède sa propre mémoire locale. Une telle distribution des pixels dans les mémoires locales des PEs revient à faire supporter entièrement au premier PE les calculs effectués sur la première colonne. Notre problème se retrouve donc dans cette machine, mais appliqué aux PEs de celle-ci, et non plus au moyen de communication entre notre mémoire et les PRR.

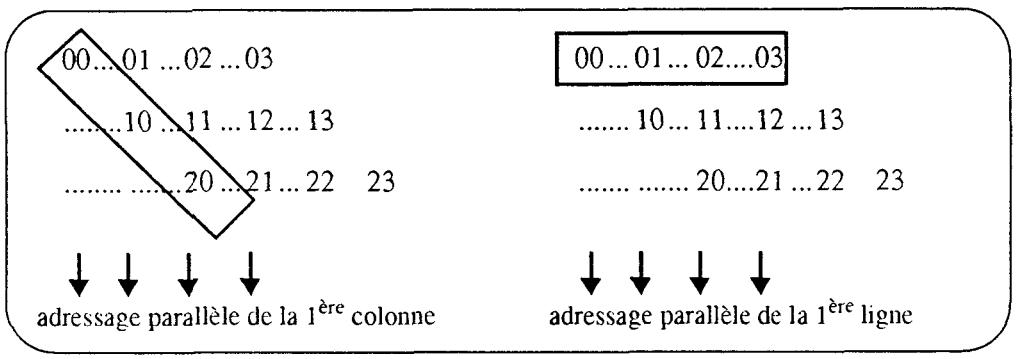


figure 13 Distribution décalée

Pour résoudre cette difficulté, l'idée consiste à décaler la distribution des pixels d'une colonne par ligne nouvelle. Comme le montre la figure 13, il est alors possible de lire soit en ligne soit en colonne, en tirant toujours partie des qxL bits du bus entre mémoire et PRR.

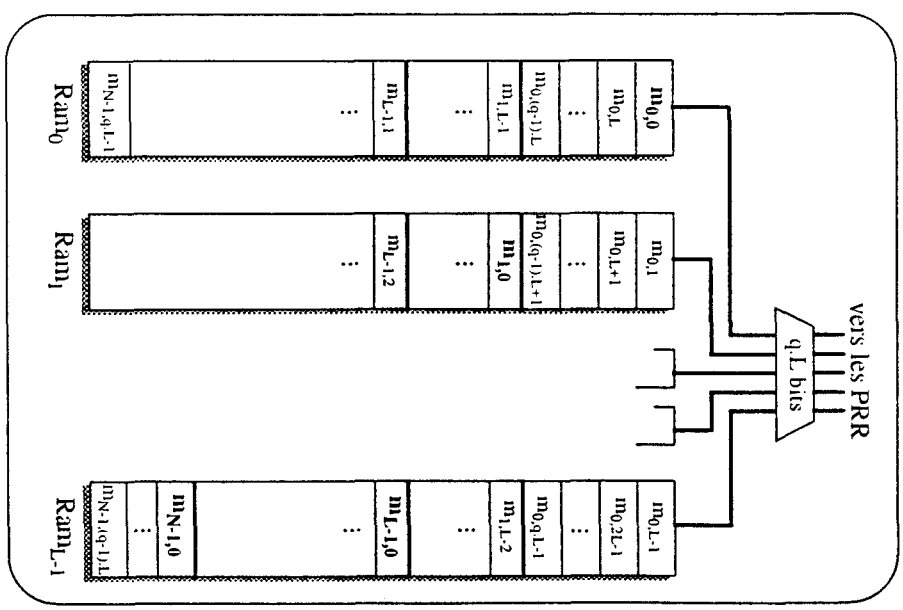


figure 14 Arrangement mémoire d'un tableau décalé

Ceci se traduit au niveau de la distribution des pixels sur les L bancs mémoire entrelacés, par la disposition en tableau décalé présentée à cette figure 14. Cette disposition est identique à la disposition en carré latin, préconisée dans les mémoires cache [MATI89-1]. Dans ces mémoires extrêmement rapides, les lectures de la mémoire centrale vers le cache s'effectuent sur plusieurs mots: 4 en général, tandis que la CPU voit la mémoire cache par un bus de largeur 1 mot. Cette différence permet d'augmenter de manière virtuelle la bande passante de la mémoire centrale.

Cette distribution permet de lire N pixels de la première colonne en réalisant N/L lectures. En effet, à chaque lecture, on récupère L mots consécutifs: par exemple $m_{0,0}$, $m_{0,1}$, $m_{0,L-1}$. Si l'on cherche à évaluer de nouveau notre critère d'efficacité, on sait qu'il faut récupérer N données, et qu'avec la disposition mémoire proposée, N/L lectures (temps = (N/L)xτ) sont nécessaires qui saisissent (N/L)xq bits. La nouvelle efficacité est donc:

$$E_2 = \frac{N \times \tau}{\left(\frac{N}{L} \times q\right) \times \left(\frac{N}{L} \times \tau\right)} = \frac{L^2}{N \times q}$$

L'observation de ce critère montre qu'au prix d'une disposition certes plus complexe des pixels, l'efficacité de ce dispositif augmente de L^2 par rapport au précédent. Cette valeur L^2 s'explique à la fois par l'absence d'information non significatives à chaque lecture (rapport de 1/L par rapport au cas précédent), mais aussi par le nombre diminué de lectures requises du fait de la disparition de lectures superflues. On observe donc que notre rapport est dangereux, puisqu'il y a corrélation entre la quantité d'information évaluée et le nombre de cycle requis pour cette évaluation.

Aussi nous proposons de nous intéresser au rapport de la quantité d'information cherchée (N pixels) sur la quantité d'information évaluée lors des lectures mémoire. Nous obtenons donc les rapports suivants (les indices sont restés les mêmes):

$E_0 = \frac{N}{N} = 1$	$E_1 = \frac{N}{N \times q} = \frac{E_0}{q}$	$E_2 = \frac{N}{\frac{N}{L} \times q} = L \times E_1 = \frac{L}{q} \times E_0$

Ce rapport exprime "mieux" la différence entre les deux méthodes, *cf.* E_1 *et* E_2. Il montre que la disposition en tableau décalé offre un gain de L par apport à la première méthode, d'autant plus important que L est important. Néanmoins ce rapport ne permet pas une évaluation juste, puisque E_0=1, et que pour L>q alors $E_2 > E_0$, ce qui ne veut rien dire. Hélas nous n'avons trouvé aucune référence à des rapports donnant l'efficacité d'une disposition mémoire.

4.2.4 les mémoires double-ports

Une mémoire double-port comporte deux ports de lecture. On peut donc adresser deux mots de cette mémoire par cycle mémoire, et récupérer sur deux ports distincts les données. L'écriture s'effectue sur un seul port, pour éviter les risques d'écriture de deux données distinctes à un même endroit de la mémoire. L'intérêt de ces mémoires double-ports apparaît selon le type d'opérations effectuées sur les données. Ainsi dans le cas d'une UAL, chaque opération requiert en général 2 lectures, les deux opérandes, pour une écriture qui concerne souvent l'une des opérandes choisies.

Pour notre processus d'extraction de segments, il faut donc analyser quels seront les besoins en lecture et écriture dans la mémoire d'images. En lecture d'abord, les PRR ont besoin de P pixels chacun par cycle de rotation. De même, le MAD d'extraction de segments peut avoir besoin de bloc de pixels. Dans le déroulement de ce processus globale d'extraction de segments, deux processus accéderont à cette mémoire d'images: les PRR du MAD de rotation, et le MAD d'extraction de segments.

Quant à l'écriture dans cette mémoire, les deux seuls utilisateurs sont les PRR, pour stocker des lignes tournées, et l'unité de contrôle du MAD de rotation pour les chargements d'images à tourner depuis la mémoire d'images de PASTIS 90. L'effort doit être porté sur l'écriture par les PRR dans leur mémoire locale, afin de ne pas ralentir leur fonctionnement.

Pour préciser, l'extraction de segments pour un angle de décomposition de $\pi/8$ nécessite la rotation de l'image traitée aux angles suivants: 0, $\pi/8$, $\pi/4$, $3\pi/8$, $\pi/2$, $5\pi/8$, $3\pi/4$, $7\pi/8$. La première écriture rendue nécessaire est le chargement de l'image, notée I_0, dans la mémoire locale du MAD de rotation, c'est ici que le processus peut être pénalisé par un moyen de communication lent: chargement de l'image à partir d'un disque ou d'une machine hôte.

Ensuite, le MAD de rotation effectue deux rotations successives: rotation d'angle $\pi/8$ de l'image I_0 qui donne l'image $I_{\pi/8}$, et rotation d'angle $\pi/8$ de l'image $I_{\pi/8}$ qui donne l'image $I_{\pi/4}$. Puis le MAD effectue une dernière rotation d'angle $-\pi/8$ de l'image I_0 qui donne l'image $I_{-\pi/8}$. Ceci nous donne le tableau suivant:

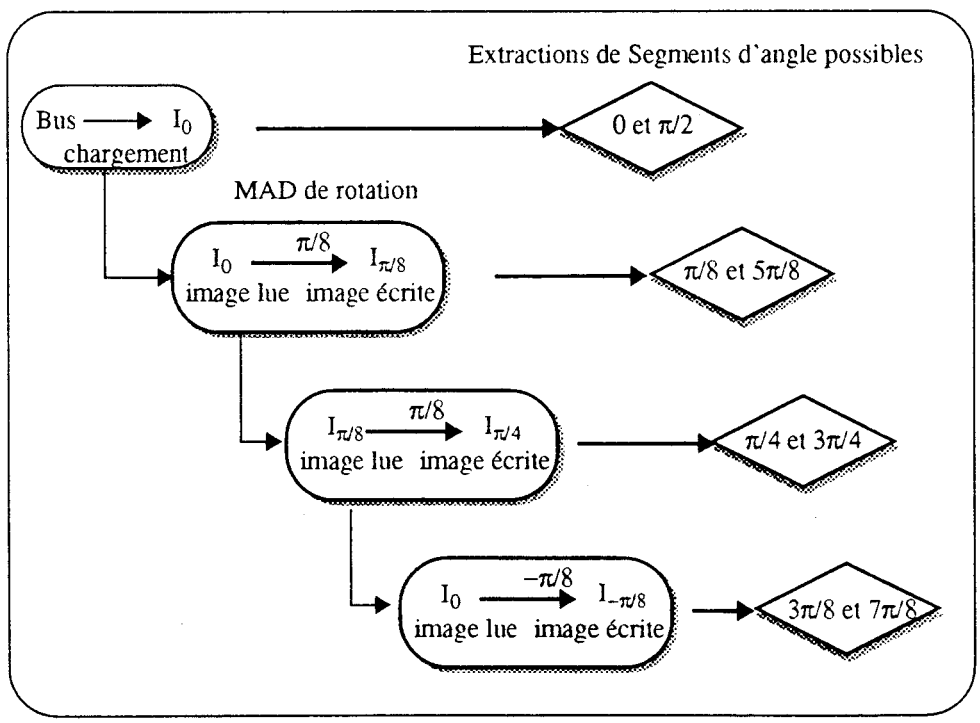


figure 15 Processus d'extraction de segments

Ce tableau montre que pour un processus d'extraction de segments avec un angle de $\pi/8$, une seule image est chargée depuis la mémoire d'images de Pastis, trois images sont écrites et trois autres sont lues (deux fois la même d'ailleurs) pour obtenir les images dont a besoin le processus d'extraction de segments.

Le MAD qui effectue cette extraction accède huit images en lecture. On voit donc que c'est sur l'accès en lecture par ce MAD d'extraction que devront porter nos efforts. Enfin ce schéma confirme que la taille de la mémoire locale doit être de l'ordre de quatre (voire six ou huit) images, l'angle $\pi/8$ étant l'un des plus utilisé pour l'extraction de segments.

4.2.5 Double bus de données

L'utilisation d'une mémoire d'images double-port apporte au processus d'extraction de segments, un plus au niveau du transfert de données. Cependant une configuration à double bus de données risque d'être coûteuse. Mais les autres solutions ne le sont-elles pas plus? D'ailleurs on peut faire le rapprochement avec CAPITAN, qui est aussi une machine à double bus de données. On peut même se poser la question de savoir si PASTIS ne pourrait pas être construite autour d'un double bus en anneau.

Ainsi, dans ce cas, les processus de rotation et d'extraction de segments pourraient se partager la mémoire d'images à grand débit. Le synoptique suivant résume cette idée:

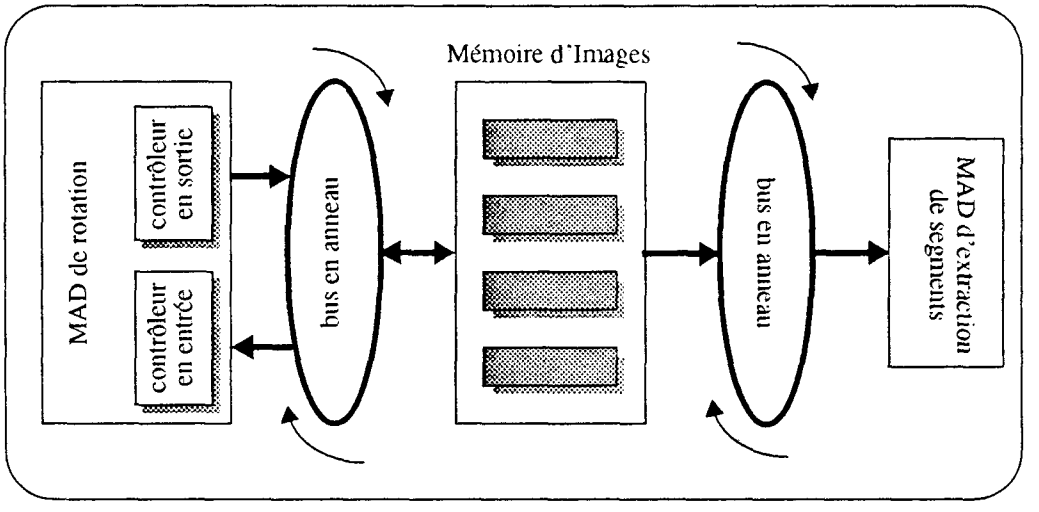


figure 16 machine à double bus de données

4.3 Gestion mémoire transparente

4.3.1 Disposition mémoire

Le but de ce paragraphe est de présenter nos idées en matière d'architecture d'une mémoire d'images. A la manière de la *staging memory* de la machine MPP, l'adresse générée par le MAD lecteur serait composée des indications suivantes: X,Y,W,H, où X,Y sont les coordonnées du point supérieur gauche d'une fenêtre de hauteur H et de largeur W.

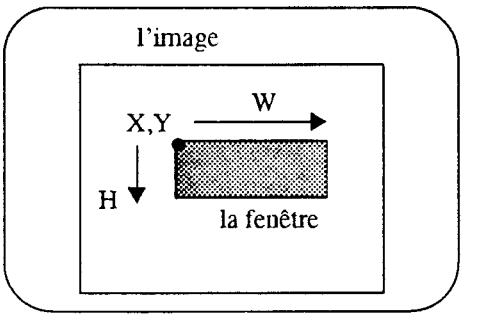


figure 17 Mode d'accès par fenêtrage

Suivant le rapport W/H, la mémoire d'images accédera les pixels en lignes ou en colonnes. Bien entendu une étude fine du fonctionnement réel de cette mémoire permettra de décider quel type d'accès privilégier, ceci en fonction de la largeur du bus de données et des valeurs W, H, et W/H. Ainsi on peut imaginer que pour W>>H, un accès en lignes serait préférable; encore que cela reste à déterminer par la multiplicité de W ou H par rapport à la largeur du bus. L'idée est d'offrir un mécanisme susceptible de tirer partie au maximum de cette largeur du bus à chaque accès mémoire.

Soit la disposition mémoire suivante:

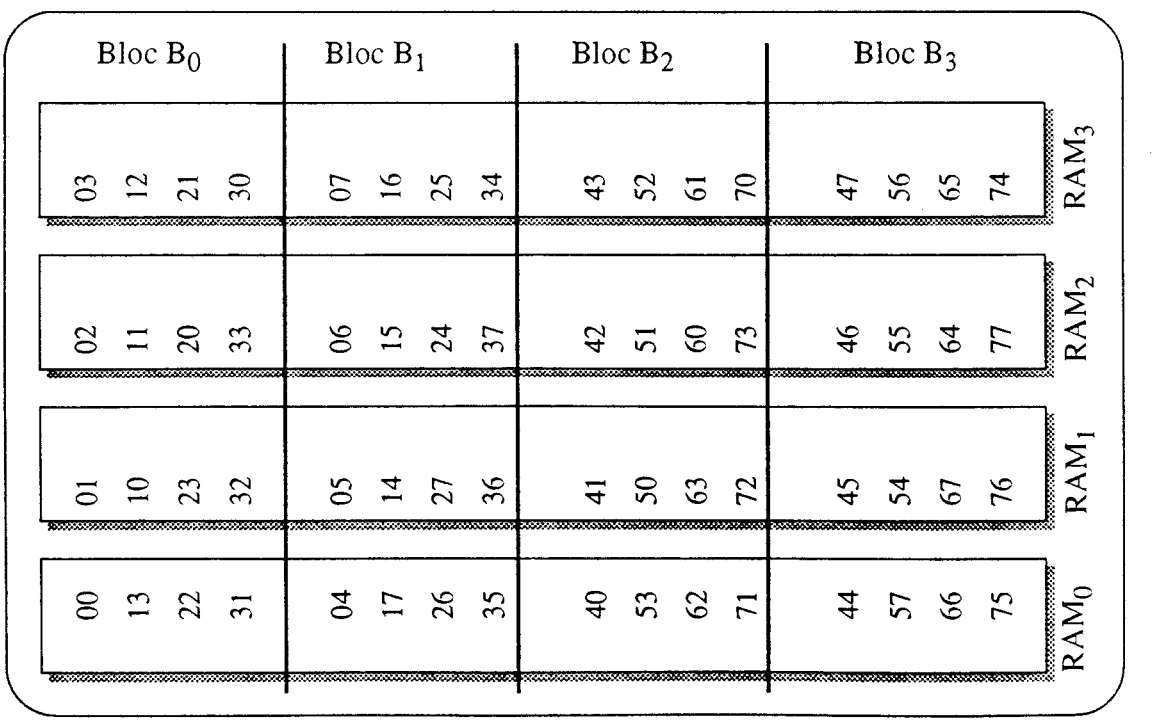


figure 18 distribution des pixels d'une image 8x8, sur quatre bancs séparés 1 bit

Cette disposition mémoire correspond à la projection sous forme de tableau décalé de l'image suivante:

image 8x8

00	01	02	03	04	05	06	07
10	11	12	13	14	15	16	17
20	21	**22**	23	24	25	26	27
30	31	**32**	33	34	35	36	37
40	41	**42**	43	44	45	46	47
50	51	**52**	53	54	55	56	57
60	61	**62**	63	64	65	66	67
70	71	**72**	73	74	75	76	77

figure 19 fenêtre accédée pour X=2, Y=2, W=1, H=6

Dans le cas de la distribution de la figure 18, il y a quatre bancs mémoire séparés. Pour profiter au maximum de la largeur du bus, on fera donc l'accès en colonne (W=1 < H=6): deux vecteurs sont extraits de la mémoire qui se composent de la sorte:

	RAM_0	RAM_1	RAM_2	RAM_3
1^{er} vecteur	22	32	42	52
2^{nd} vecteur	62	72	x	x

Au passage le lecteur remarquera que les deux scalaires 22 et 32 se trouvent dans le bloc B_0, conformément à la figure 18, tandis que les scalaires 42 et 52 se trouvent dans le bloc B_2. L'image a en effet été découpée en blocs de 4x4 pixels, de la manière suivante:

BLOC B0 **BLOC B1**

00	01	02	03	04	05	06	07
10	11	12	13	14	15	16	17
20	21	22	23	24	25	26	27
30	31	32	33	34	35	36	37
40	41	42	43	44	45	46	47
50	51	52	53	54	55	56	57
60	61	62	63	64	65	66	67
70	71	72	73	74	75	76	77

BLOC B2 **BLOC B3**

figure 20 découpage de l'image 8x8 en blocs de 16 pixels

Adresses logique et physique

Nous voyons donc qu'il faut fournir une succession d'adresses physiques à partir de l'adresse logique (X,Y,W,H). Différentes pour chacune des RAMs auxquelles elles sont données, ces adresses physiques comportent deux champs: le numéro du bloc et le numéro du pixel dans ce bloc. Comme ces calculs d'adresses effectives seront effectués par des additionneurs dont il faut limiter le nombre de bits en entrée par souci des performances, nous avons donc structuré une image de taille N^2 en K^2 blocs de P^2 pixels chacun, de la manière suivante:

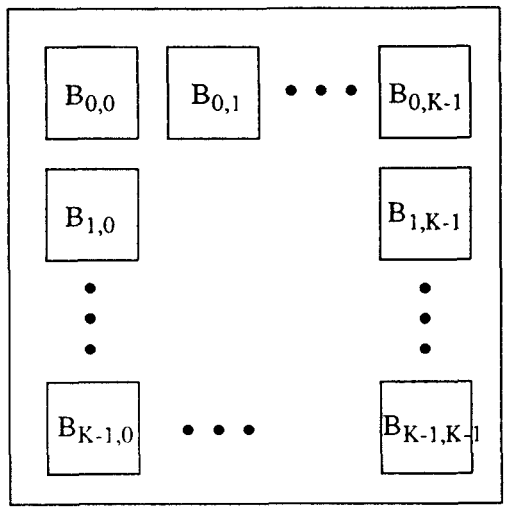


figure 21 image divisée en K^2 blocs de P^2 pixels

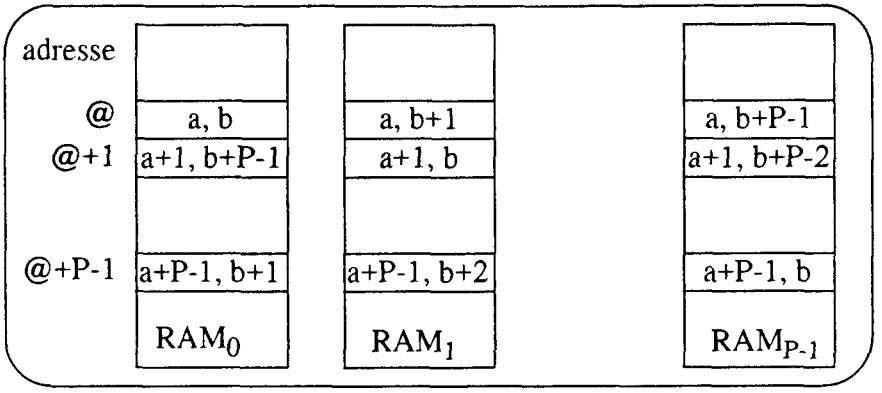
Le contenu du bloc de pixels $B_{i,j}$ où *i=0,...,K-1* et *j=0,...,K-1*, est le suivant (on pose a=i.P et b=j.P, et l'on donne dans la figure la position des pixels dans l'image: ligne, colonne):

P pixels en largeur

P pixels en hauteur

a, b	a, b+1		a, b+P-1
a+1, b	a+1, b+1		a+1, b+P-1
a+P-1, b	a+P-1, b+1		a+P-1, b+P-1

Quant à la projection de ce bloc sur la mémoire d'images qui comporte *P* RAMs 1 bit, elle se fait de la manière suivante:



L'adresse @ est donné par @=i.P.K + j.P, puisque chaque bloc comporte P pixels et qu'une ligne de blocs comporte K blocs. Cette adresse est celle du bloc $B_{i,j}$. Par la suite, nous notons la division entière par l'opérateur /, et le reste par l'opérateur %.

Coordonnées dans l'image et adresse physique

Soit un pixel de coordonnées (ligne, colonne) dans l'image, à quelle adresse physique et dans quelle RAM est-il stocké?

Soit *i=ligne/P* et *j=colonne/*P, alors ce pixel appartient au bloc $B_{i,j}$ dans l'image. Soit δi=ligne%P, alors l'adresse @ du pixel est la suivante: *@=@(Bloc B_{ij}) + déplacement(δi)*. La disposition choisie donne donc **@ = i.P.K + j.P +** δi. Ceci correspond à la ligne physique où se trouve stocké le pixel.

La RAM dans laquelle il se trouve dépend du déplacement dû à la structure en tableau décalé. On a donc le numéro de la RAM où se trouve le pixel: **num(RAM) = (δi + δj) modulo P**, où δj=colonne%P.

Ce dernier résultat prouve que quelques soient les coordonnées d'un pixel de départ (ligne, colonne), les P pixels suivants en ligne ou en colonne se trouvent stockés dans des RAMs de numéros différents: numéros croissant ou décroissant suivant le sens de la suite des pixels.

4.3.2 Unité de génération d'adresses physiques

A partir de cette disposition mémoire, nous avons imaginé l'unité de génération d'adresses physiques, nommée GAMI. Chaque RAM de mots de 1 bit composant la mémoire image de la machine, possède sa propre unité GAMI. Les échanges se font donc *via* celle-ci, dont le rôle est de transformer une adresse logique (coordonnées du pixel) en une adresse physique. Nous allons voir comment s'obtient cette translation d'adresses.

Soient X et Y la colonne et la ligne du pixel de référence de la fenêtre de largeur W et de hauteur H à saisir. On suppose comme à la figure 21, que notre image est divisée en K^2 blocs de P^2 pixels chacun. L'adresse délivrée à une RAM doit donc comporter trois champs: un champ pour le choix de la ligne de blocs, un champ pour le bloc dans cette ligne, et enfin l'adresse du pixel dans ce bloc.

Le format de cette adresse est donc le suivant:

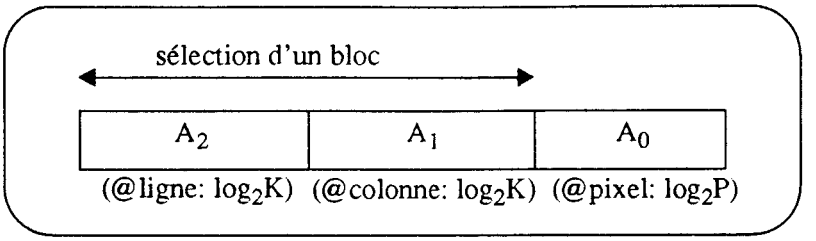


figure 22 format d'une adresse physique

La sélection du bloc se fait sur les champs ligne et colonne avec $\log_2 K$ bits, puisqu'il y a K^2 blocs découpés. On pourrait ajouter sur le champ ligne un nombre plus important de bits qui permettraient la sélection d'un plan de bits, i.e. d'une image, dans le cas où plusieurs sont stockées.

D'après les calculs précédents, il est évident que:

$$A_0 = (Y + X) \% P$$
$$A_1 = X / P$$
$$A_2 = Y / P$$

si la donnée (X,Y) correspond au pixel choisi. Dans notre cas, nous n'accédons pas pixel par pixel, mais par bloc de P pixels consécutifs, en ligne ou en colonne. Il faut donc à partir de la donnée d'un pixel (X,Y) calculer les adresses des P pixels en ligne ou en colonne. Les équations pour A_1 et A_2 démontrent que seuls les $\log_2 K$ bits de poids fort seront utilisés pour le choix du bloc: donc l'adresse réelle est simple à calculer à partir de X et Y.

Le choix de l'accès en ligne ou en colonne dépend du *ratio* W/H. Ce choix est donc établit par le contrôleur de cette mémoire. Soit C(W/H) le choix effectué: C(W/H)=0 implique un accès en colonne, C(W/H)=1 un accès en ligne. Dans le cas d'un accès en colonne (attention: les P pixels se trouvent sur P lignes consécutives dans la même colonne!), par exemple, la question est de savoir si les P pixels à accéder se trouvent dans le même bloc ou non. Ce cas est réalisé à la condition que Y%P soit nul (=0). Dans les autres cas, une partie des pixels se trouveront dans le bloc d'adresse (Y/P,X/P), l'autre partie dans le bloc d'adresse (Y+1/P,X/P). Pour les adresses, on a deux possibilités:

$$A_2 = Y/P \text{ et } A_1 = X/P$$
$$A_2 = Y/P+1 \text{ et } A_1 = X/P$$

Le choix sera fait suivant l'emplacement de notre fenêtre ligne sur le découpage en blocs de l'image. Si cette fenêtre de P pixels de haut, nous considérons toujours un accès en colonne, se trouve sur un seul bloc, la condition Y%P=0 est réalisée. Si $Y\%P=v$, alors on ne peut prendre dans le bloc d'adresse ($A_2 = Y/P$ et $A_1 = X/P$) que les v premiers pixels, et dans le bloc d'adresse ($A_2 = Y/P+1$ et $A_1 = X/P$) les P-v autres pixels.

La difficulté est de savoir, pour une RAM donné, par exemple la $i^{ème}$, quelle adresse de bloc délivrer? Soit $A_0 = (Y + X) \% P$, et i le numéro de la RAM concerné, i=0...P-1, alors on montre que pour A_0=i l'adresse du bloc est (A_2=Y/P et A_1=X/P), tandis que pour A_0>i l'adresse du bloc est ($A_2 = Y/P+1$ et A_1=X/P).

La figure 23 propose un schéma synoptique d'une telle unité. Les incrémenteurs permettent de faire fonctionner le GAMI en circuit DMA pour les accès au bloc complet: W.H pixels. Nous n'avons pas poussé plus loin la conception en ce domaine.

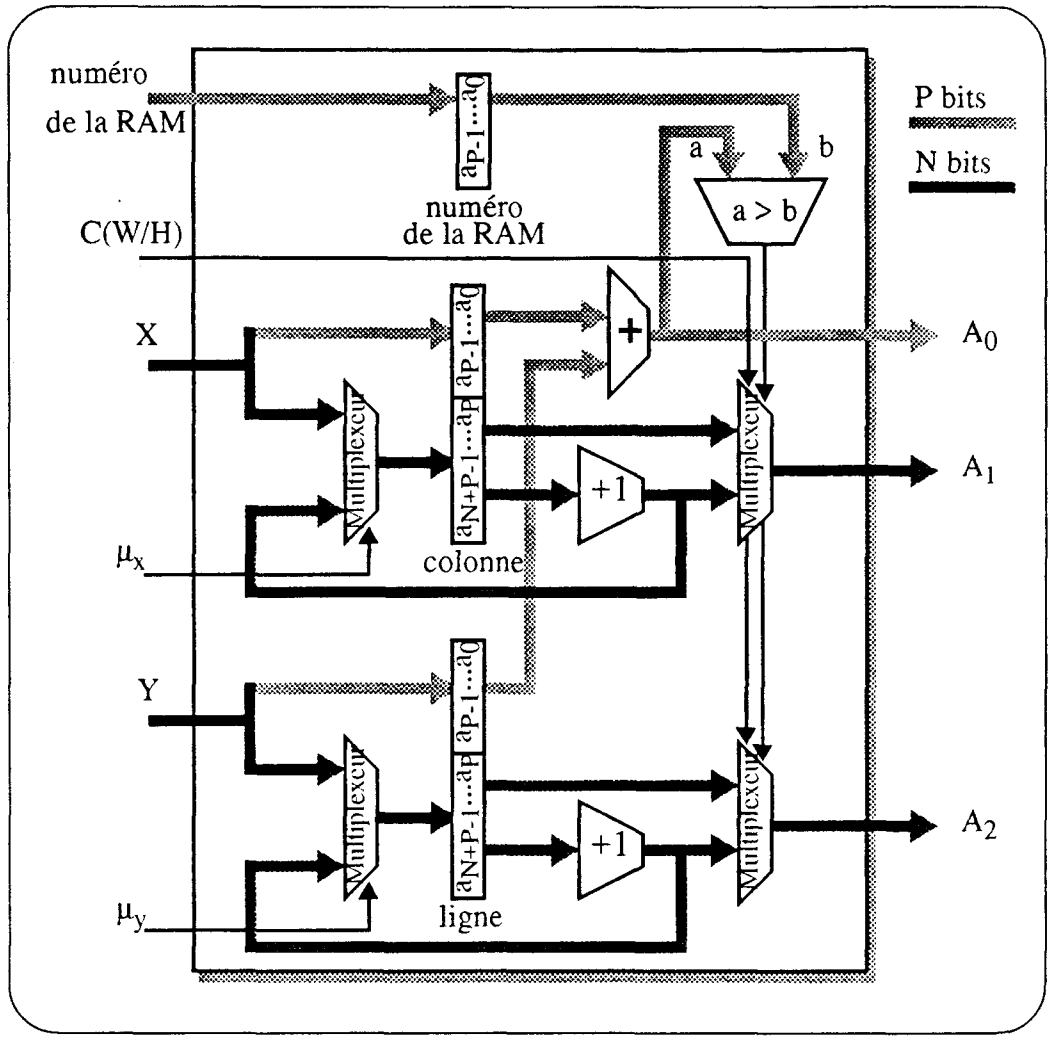


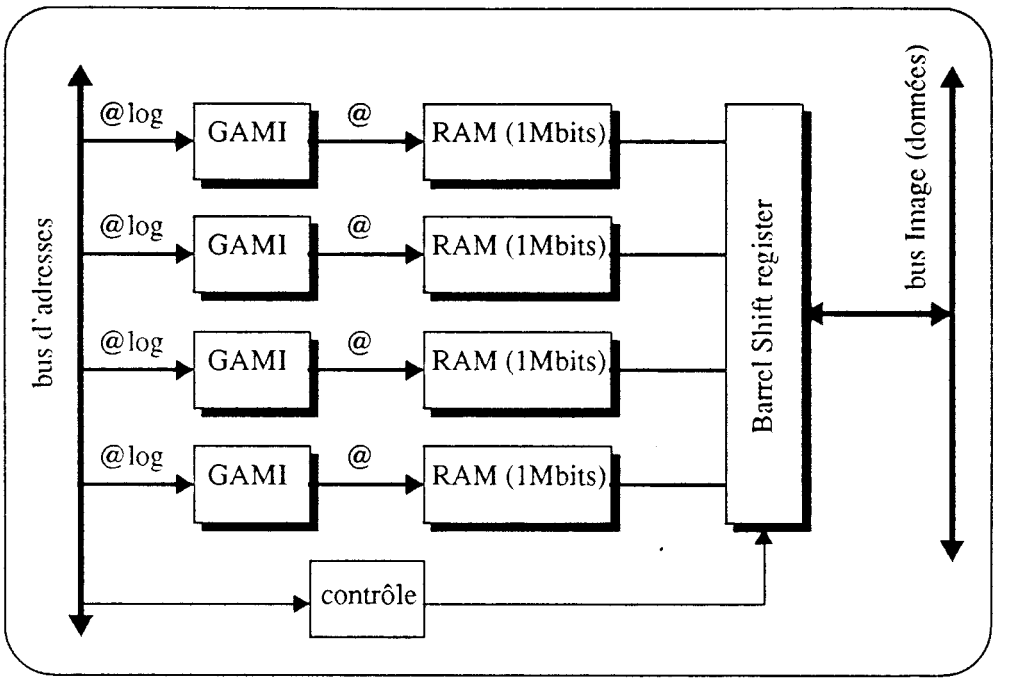
figure 23 GAMI, générateur d'adresses pour la mémoire d'images

4.3.3 Schéma global de la mémoire d'images

A l'aide de ce générateur d'adresses, il est facile de réaliser une mémoire d'images. Chaque générateur GAMI adresse des mots de 1 bit, et donc une mémoire de N mots de 1 bits, Nx1 bits. Sur une machine possédant un bus de largeur L bits, pour construire une mémoire d'images de T pixels, où T serait un multiple de la taille de l'image de référence, il faut donc L RAMs 1 bit, d'une capacité de T/L bits chacune. De plus, associé à chacune des RAMs, il faut un générateur GAMI; c'est à dire L GAMIs et L RAMs 1 bit en tout.

Remarquons que l'utilisation de la disposition mémoire permet d'utiliser des mémoires 1 bit, dont on sait qu'elles évoluent comme une référence du "state of art" dans le domaine des mémoires. Un accroissement de la taille de l'espace mémoire ne posera pas d'autres difficultés que celle de modifier les sorties adresses du GAMI. La taille de l'adresse A_2 du GAMI pourrait être importante, sans nécessairement utiliser tous les bits de cette adresse.

On obtient donc le schéma suivant:



Le *barrel shift register* est utilisé pour repositionner les pixels provenant du bus Image en face de la RAM qui les contient.

5 Conclusion

Les différentes architectures présentées étayent nos arguments en faveur d'une machine temps réel pour la vision. Le processeur de rotation a été conçu et réalisé complètement en ce qui concerne les circuits ASICs; seules les simulations de ces circuits et la réalisation d'une carte les intégrant permettraient une conclusion certaine quant à la complexité du traitement. Pour le MAD d'extraction de segments et la mémoire d'images, leur description offrent un prolongement architectural à ce travail dont on aurait pu apprécier à terme l'intérêt si une suite s'était engagée.

Bien que tel ne fût pas le cas, la conception à demi achevée aura eu un impact positif sur l'ensemble de ces travaux. En démontrant la consistance d'une approche algorithmique liée aux considérations matérielles, nous avons tenté de trouver un modèle satisfaisant pour la vectorisation rapide d'images de grande taille.

Références Bibliographiques du Chapitre 3

[ARAB87-1] Arabnia HR, Oliver MA,
"Arbitrary rotation of raster images with SIMD machine architectures",
Computer Graphics Forum 6, pp.3-12, 1987

[ARAB87-2] Arabnia HR, Oliver MA,
"A transputer network for the arbitrary rotation of digitised images",
The Computer Journal, 30, no 5, 1987

[DELE89] Deleu J.
"Deux propositions architecturales d'un processeur rapide de rotation",
Publication interne LIFL, no 75, 1989

[KUCK68] Kuck D.J.,
"Illiac IV Software and Application Programming",
IEEE Transactions on Computers, Vol. C-17, N.8, August 1968.

[MATI89-1] Matick R.E.,
"Functionnal cache chip for improved system performance",
IBM J.Res.Devel., Vol 33, N. 1, Jan. 89

[MATI89-2] Matick R. & al,
"Architecture, design, and operating characteristics of a 12ns CMOS functional cache chip",
IBM J.Res.Devel., Vol 33, N. 5, Sept.89

[PRC89] Compte rendu des Premières Journées du PRC Architecture de Machines Nouvelles, Grenoble, les 8 & 9 Novembre 1989

[THOR88] Thorpe Sj,
"Traitements d'images chez l'homme",
TSI, vol. 7, No 6, pp.517-525, 1988

"le linéaire sinon rien"
apophtegme moderne dont Max Dauchet
ne saurait nier la paternité...

Conclusion

Parce que les outils manquaient, la vision ne s'est ouvert aux chercheurs que depuis un petit siècle. Ceux-ci se retrouvent face à un modèle fonctionnel dont les réalisations humaines et animales procèdent d'une lente évolution naturelle, toujours en quête d'un équilibre perfectible entre l'acuité des différents sens impliqués chez ces êtres et leurs fonctions vitales. Pourrions-nous en un siècle étudier, appréhender et embrasser un tel mécanisme?

Si une réponse négative s'impose *de facto*, alors il nous faut admettre que la plus petite parcelle de connaissances est encore bonne à investir. Pour la vectorisation d'images, l'approximation polygonale est sans conteste la meilleure solution qui n'a d'égale que sa complexité. Puisque chacun s'accorderait sans doute pour lui attribuer une forme algorithmique *naturelle* du problème de la vectorisation d'images, les recherches en cours n'ont qu'un objectif: lui trouver des pré-traitements qui en améliorent la réalisation en diminuant sa complexité.

Par un chassé-croisé de l'algorithme vers l'architecture, ces pré-traitements ont été détaillés: transformée de Hough, *template matching*, dilatation et érosion morphologiques, ou approche cellulaire orientée. La méthode proposée dans cette thèse n'a rien d'original aux regards des autres. Elle se rattache d'ailleurs à chacune d'entre elles, signe prouvant à l'occasion qu'un domaine mal cerné peut s'offrir beaucoup de méthodes différentes.

L'intérêt de notre proposition réside dans le lien qu'elle génère avec l'architecture. Ce lien se renforce quand l'architecture laisse présager une réalisation en $O(n)$ de la vectorisation d'images contenant n^2 pixels. La cohérence établie en filigrane entre les hypothèses d'approximation faites sur le traitement et ses incidences quant à la conception du processeur de rotation rapide par exemple, confirme notre opinion sur la nécessaire adéquation des études algorithmiques et architecturales.

A une époque où le processeur dédié devient une nécessité, elle justifie à elle seule l'ensemble de ce travail, depuis la description d'un nouvel algorithme de vectorisation jusqu'à notre réalisation de circuits électroniques.

Annexe

```
/*************************************************************************/
/*                                                                       */
/*      Programme de Simulation du Fonctionnement du                     */
/*                                                                       */
/*              Processeur de Rotation Rapide                            */
/*                                                                       */
/*************************************************************************/
/*                                                                       */

#include "imag.h"       /* fichier contenant les cnstes pour affichage  */


/*                                                                       */
/*                                                                       */

#define alpha   (-PI/8.0)       /* angle de rotation */



/*                                                                       */
/*************************************************************************/
/*                                                                       */
/*      PREMIER ETAGE du pipeline:                                       */
/*                                                                       */
/*                              Unite de Rotation                        */
/*                                                                       */
/*************************************************************************/
/*                                                                       */
/*                                                                       */
/*************************************************************************/
/*                                                                       */
/*      Definition des variables:       (1) memoire de ligne tournee     */
/*                                      (2) registre MASQUE              */
/*                                      (3) reseau de recouvrement OR    */
/*              N pixels en entree      (4) registre de SORTIE           */
/*                  ||                                                   */
/*              recouvrt OR                                              */
/*                  ||                  -------------                    */
/*              -------------           |           |                    */
/*              | MASQUE    |           | MEMOIRE   |                    */
/*              -------------           -------------                    */
/*                  ||                      ||                           */
/*                  ----------||  ||-----------                          */
/*                            ||  ||                                     */
/*                          portes ET                                    */
/*                            ||                                         */
/*                           -----------                                 */
/*                          | SORTIE    |                                */
/*                           ----------                                  */
/*                            ||                                         */
/*                      vers le second etage                             */
/*                                                                       */
/*                                                                       */
/*************************************************************************/
/*                                                                       */
```

```
int     MEMOIRE[N][N],          /* la memoire de ligne tournee          */
        MASQUE[N],              /* le registre masque                   */
        TABREC[N],              /* le reseau de recouvrement            */
        SORTIE[N],              /* le registre de sortie                */

        LIG0[N], LIG1[N];       /* tableaux temporels de pixels         */


float   Cosinus,                /* les valeurs cos(alpha) et sin(alpha) */
        Sinus;                  /* sont stockees dans ces variables     */



/*                                                                      */
/************************************************************************/
/*              affichage des images resultantes                        */
/*                                                  (sans interet...)   */
/************************************************************************/
/*                                                                      */


        afficher(image)
        int     *image;
        {
           register     int     lig,col;

           for (lig=N+irint(N*Sinus); lig<N; lig++)
              for (col=0; col<N; col++)
                  pw_rop(pw_proc,col*K,lig*K,K,K,PIX_SRC,
                         (*image++ ? PAVE_NOIR : PAVE_BORD),0,0);
           return;
        }


        char_ligne(ligne)
        int     *ligne;
        {
           memcpy(&MASQUE[0],ligne,N*sizeof(int));
        }


        sort_ligne(ligne,depx,depy)     /*on travaille avec Imagebis[][]*/
        int     *ligne,depx,depy;
        {
           int  delta, *tab;

           if ((depy<0)||(depy>=N)) return;
           delta = ((depx > 0) ? N-depx : N+depx);
           while (depx < 0) {depx++;ligne++;}
           tab=&Imagebis[depy][depx];
           while (delta--) *tab++ = *ligne++;
        }
```

```
/*                                                                          */
/****************************************************************************/
/*                                                                          */
/*      Initialisation des donnees du premier etage:                        */
/*                                                                          */
/*                      (1) la memoire de ligne tournee est chargee         */
/*                      (2) le reseau de recouvrement est calcule           */
/*                                                                          */
/****************************************************************************/
/*                                                                          */

        remp_memoire()          /* remplit la memoire de ligne tournee  */
        { /* debut de remp_memoire() */

                int     lig, col, val_lig;

                for (col=0; col<N; col++)
                   {
                   val_lig = irint((col-N/2)*Sinus) - irint(-N/2*Sinus);
                                /* calcul de la ligne associee a 'col'  */

                   for (lig=0; lig<N; lig++)
                        {
                        if (val_lig == lig)     MEMOIRE[lig][col] = 1;
                        else                    MEMOIRE[lig][col] = 0;
                        }
                   }

        } /* fin de remp_memoire() */

/*                                                                          */
/****************************************************************************/
/*                                                                          */

        remp_tabrec()           /* remplit le tableau de recouvrement   */
        { /* debut de remp_tabrec() */

                int     col, nouv_col, *tab, col_prec;

                col_prec = 0;
                for (col=0, tab=&TABREC[0]; col<N; col++) {

                    nouv_col = irint((col-N/2)*Cosinus)-irint(-N/2*Cosinus);
                                /* calcul de la nouvelle colonne */

                    if ( (col!=0)  &&  (nouv_col == col_prec) )
                                /* test si la valeur 'nouv_col' recouvre*/
                                /* la valeur precedente: 'col_prec'     */

                          {  tab--;    *tab++ =1; }
                                /* ds ce cas, TABREC[col_prec] = 1      */


                   *tab++ =0;                   /* on passe a la suite..*/
                   col_prec = nouv_col;

                } /* fin de boucle FOR */

        } /* fin de remp_tabrec() */
```

```
/*                                                                      */
/************************************************************************/
/*                                                                      */
/*      Fonctions du premier etage:                                     */
/*                                                                      */
/*                      (1) la gestion du recouvrement: recouvre()      */
/*                      (2) le chargmt d'une ligne:     ent_bitmap()    */
/*                                                                      */
/************************************************************************/
/*                                                                      */

        recouvre()              /* gere le recouvrement, par portes OU  */
        { /* debut de recouvre() */

                int     *tab_or, *lig_avt, *lig_apr, nbre_col;


                tab_or  = &TABREC[0];
                lig_avt = &LIG0[0];     /* pixels qui sont a l'entree   */
                lig_apr = &LIG1[0];     /* pixels qui sont en sortie    */
                nbre_col= N;


                while (nbre_col > 0) {
                    if (*tab_or++ == 1) {
                                        /* le pixel actuel doit etre    */
                                        /* recouvert avec le suivant    */
                        *lig_apr = *lig_avt++;
                        nbre_col--;
                                        /* on donne la valeur au pixel  */
                        while (*tab_or++ == 1) {
                           *lig_apr |= *lig_avt++;
                           nbre_col--;
                           }
                                /* le WHILE sert au cas ou plusieurs    */
                                /* pixels successifs se recouvrent (rare*/
                        *lig_apr |= *lig_avt++;
                                /* le OU permet le recouvrt avec suivant*/
                        lig_apr++;
                                /* on avance le pointeur: lig_apr       */
                        nbre_col--;

                    } else {
                        /* il n'y a pas de recouvrement, rien a changer*/
                        *lig_apr++ = *lig_avt++;
                        nbre_col--;
                    }
                } /*fin du WHILE*/
        } /* fin de recouvre() */
```

```
/*                                                                          */
/****************************************************************************/
/*                                                                          */


        int     *ent_bitmap(num)
        int     num;
        { /* debut de *ent_bitmap() */
                        /* cette fonction renvoie un pointeur sur LIG1, */
                        /* qui contient la NUMieme ligne a charger ds le*/
                        /* bitmap.                                      */


                int     nbre_col, *tab, *lig;


                memset(LIG0, 0, N*sizeof(int));
                                        /* mise a zero de LIG0[]        */
                lig = &LIG0[0];
                tab = &MEMOIRE[num][0];
                for (nbre_col=N; nbre_col--; )
                     *lig++ = *tab++;
                                /* chargement de la ligne tournee (num) */

                lig = &LIG0[0];
                tab = &MASQUE[0];
                for (nbre_col=N; nbre_col--; lig++)
                     *lig &= *tab++;
                                /*comparaison AND avec le registre MASQUE*/

                recouvre();
                                /* appel de recouvre() qui fait le recou*/
                                /* vrement de LIG0[], en LIG1[]         */

                return(&LIG1[0]);
                                /* on retourne LIG1[], qui contient les */
                                /* pixels pour l'entree du Bitmap       */
        } /* fin de *ent_bit_map() */
```

```
/*                                                                          */
/****************************************************************************/
/*                                                                          */
/*      DEUXIEME ETAGE du pipeline:                                         */
/*                                                                          */
/*                              Unite de translation, ou BITMAP buffer      */
/*                                                                          */
/****************************************************************************/
/*                                                                          */


#define LAR_BM  N
#define PRF_BM  N


/*                                                                          */
/* Remarque: PRF_BM definit la hauteur du BitMap buffer, que nous avons     */
/*           fixee a N pour la raison simple d'allocation d'un tableau.     */
/*           En fonctionnement reel, cette hauteur sera ramenee a sa        */
/*           valeur donnee par: PRF_BM = N*Sinus(alpha).                    */
/*                                                                          */
/*           Quant a LAR_BM, elle definit la largeur du BitMap buffer,      */
/*           qui est donc de N par defaut ici. En realite: on a la          */
/*           valeur suivante: LAR_BM = N*cosinus(alpha).                    */
/*                                                                          */


int     referx_bp, refery_bp,
                /* les anciennes coord. du point de reference du bitmap */

        referx_li, refery_li,
                /* les nouvelles coord. du point de reference du bitmap,*/
                /* au chargement d'une nouvelle ligne tournee           */

        CELL_TRANS[PRF_BM][LAR_BM],
                        /* les cellules de transfert du bit-map buffer  */

        CELL_MEMOI[PRF_BM][LAR_BM];
                        /* les cellules memoire du bit-map buffer       */

/*                                                                          */
/****************************************************************************/
/*                                                                          */
/*      les fonctions de cette unite:                                       */
/*                                                                          */
/*      - calcul du nouveau point de reference de l'unite                   */
/*      -                                                                   */
/*      -                                                                   */
/*                                                                          */
/****************************************************************************/
/*                                                                          */
```

```
        point_dep(x, y)
        int     x, y;
        { /* debut de point_dep() */

                        /*cette fonction calcule le nouveau point de    */
                        /* reference de la ligne a entrer (x==0)        */

                referx_li = irint((x-N/2)*Cosinus)+ N/2
                                + irint(-(y-N/2)*Sinus);
                refery_li = irint((x-N/2)*Sinus)
                                + irint((y-N/2)*(Cosinus-1))+y;

        } /* fin de point_dep() */


/*                                                                      */
/************************************************************************/
/*                                                                      */

        int     *oter_ligne(nouv_lig)
        int     nouv_lig;
        { /* debut de *sort_ligne() */
                        /* cette fonction sort une ligne du BitMap, au  */
                        /* debut de chaque rotation de ligne            */

                int     *lig_sortie, *temp0, *temp1;
                int     lig, col;

                int     nouv_col = 0;
                        /* (si #0) indique l'indice de la colonne dans  */
                        /* la version parallele de l'architecture       */


                /* calcul des nouvelles coordonnees                     */
                referx_li = irint((nouv_col - N/2) * Cosinus) + N/2
                            + irint(-(nouv_lig - N/2) * Sinus);

                refery_li = irint((nouv_col - N/2) * Sinus)
                            + irint((nouv_lig - N/2) * (Cosinus-1)) + nouv_lig;


                /* extraction de la derniere ligne stockee ds le bit-map*/
                lig_sortie  =   &LIG0[0];
                temp0       =   &CELL_MEMOI[PRF_BM - 1][0];
                for (col=0; col< N; col++)
                    *lig_sortie++ = *temp0++;
                        /* c'est cette ligne que renvoie la fonction    */


                                /* decalage Nord->Sud du bit-map?       */
                if (refery_li != refery_bp) {
                    temp0  =    &CELL_MEMOI[PRF_BM - 1][N-1];
                    temp1  =    &CELL_MEMOI[PRF_BM - 2][N-1];

                    for (lig = (PRF_BM - 1)*N; lig--;)
                        *temp0-- = *temp1--;
                                /* decalage vers le bas des cellules mem*/
                    for (col = N; col--;)
                        *temp0-- = 0;
```

```
                                        /* 00..00 sur la premiere ligne */
                }           /* fin decalage Nord->Sud...*/


                            /* decalage Ouest->Est du bit-map?      */
            if (referx_li != referx_bp) {
                temp0  =    &CELL_MEMOI[PRF_BM - 1][N-1];
                temp1  =    temp0 - 1;   /*attention: sizeof(int)==1*/

                for (lig = PRF_BM - 1; lig--; ) {
                    for (col = N-1; col--;)
                        *temp0-- = *temp1--;
                            /* decale 1 bit a droite sur N-1 bits   */
                    temp1--;
                    *temp0-- = 0;       /* le bit le + a gauche = 0 */
                    }
                }           /* fin decalage Ouest->Est */


            referx_bp = referx_li;
            refery_bp = refery_li;
            /* modification des coordonnees de reference du bit-map */

            return(&LIG0[0]);
                    /* on retourne la ligne extraite pour affichage */
        } /* fin de *sort_ligne() */


/*                                                                      */
/************************************************************************/
/*                                                                      */


        transl_bas(lig)         /*decale vers le bas CellTrans, et charge*/
        int     *lig;           /*lig sur la premiere ligne*/
        {
                int     y, *tabi, *tabf;

                tabf = &CELL_TRANS[PRF_BM -1][LAR_BM -1];
                tabi = &CELL_TRANS[PRF_BM -2][LAR_BM -1];
                for (y = LAR_BM*(PRF_BM - 1); y--;)
                        *tabf-- = *tabi--;
                        /* copie des MAR-1 lignes du bas decalees       */

                for (tabf = &CELL_TRANS[0][0], y = LAR_BM; y--;)
                     *tabf++ = *lig++;
                        /* entree de 'lig' ds la matrices de transfert  */
        }


/*                                                                      */
/************************************************************************/
/*                                                                      */


        stocker_ligne(delig)
        int     delig;
        {               /* translation ds les cellules de transfert,puis*/
```

```
                        /* stockage ds les cellules memoire a la fin    */

            int  ligne, *lig, *tab;
            int  vide[LAR_BM];

            char_ligne(&Image[delig][0]);
                        /* chargemt de la ligne a tourner dans MASQUE   */

            memset(CELL_TRANS, 0, PRF_BM * LAR_BM * sizeof(int));

            for (ligne=0; ligne < (-N*Sinus+1);ligne++) {
                        /* appel d'une fonction du 1er etage... */
                lig = ent_bitmap(-ligne);
                        /* on recupere une nouvelle ligne via   */
                        /* la memoire de ligne et le reg MASQUE */
                transl_bas(lig);
                        /* on decale vers le bas les cellules   */
                }

            memset(vide, 0, LAR_BM * sizeof(int));
            while (ligne++ < PRF_BM)
               transl_bas(vide);
                        /* on met '00..00' en entree du bitmap  */
                        /* et on decale les cell de transf...   */

            tab = &CELL_MEMOI[0][0];
            lig = &CELL_TRANS[0][0];
            for (ligne = PRF_BM * LAR_BM; ligne--; tab++)
                 *tab |= *lig++;
                        /* stockage du contenu des cellules de  */
                        /* transf ds les cellules memoire       */
        }
^L


/************************/
/* controle global      */
/************************/

/*fonct*/
cadence()
{
int     y,stx,sty;
point_dep(0,N-1);
referx_bp = referx_li;
refery_bp = refery_li;
for (y=N;y--;) {
   printf("ca tourne, ligne:%3d\n",y);
   stx=referx_bp;sty=refery_bp;
   point_dep(0,y);
   sort_ligne(oter_ligne(),stx,sty);
   stocker_ligne(y);
   afficher(&CELL_MEMOI[N+irint(N*Sinus)][0]);
   }
for (y=-N*Sinus;y--;) {
   printf("ca se termine, ligne:%3d\n",y);
   sort_ligne(oter_ligne(),stx,refery_li--);
   afficher(&CELL_MEMOI[N+irint(N*Sinus)][0]);
   }
```

```
}


simulfrp()      /*on arrive avec Image[][], on repart*/
{               /*avec Imagebis[][]*/
        int i,*tab;
        Cosinus=cos(alpha);
        Sinus=sin(alpha);

        init_memoire();
        remp_tabrec();
        memset(CELL_MEMOI, 0, PRF_BM * LAR_BM * sizeof(int));
        cadence();

}
```