USTL FLANDRES ARTOIS

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

LABORATOIRE D'INFORMATIQUE FONDAMENTALE DE LILLE

N° d'ordre : 368

THESE

présentée à

L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE FLANDRES ARTOIS

pour obtenir le titre de

DOCTEUR en INFORMATIQUE

par

LEPRETRE Eric

ALGORITHMES PARALLELES ET ARCHITECTURES CELLULAIRES POUR LA SYNTHESE D'IMAGES

Thèse soutenue le 22 juin 1989 devant la commission d'Examen

Membres du jury

B. TOURSEL	Président
M. LUCAS	Rapporteur
C. PUECH	Rapporteur
M. MERIAUX	Directeur de thèse
V. CORDONNIER	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

REMERCIEMENTS

Je tiens à remercier ici :

M[r] Bernard TOURSEL, Professeur à l'Ecole Universitaire D'Ingénieurs de Lille, qui a bien voulu présider ce jury,

Messieurs Michel LUCAS, Professeur à l'Ecole Nationale Supérieure Mécanique de Nantes et Claude PUECH, Professeur à l'Ecole Normale Supérieure, rapporteurs de cette thèse, qui m'ont fait l'honneur de consacrer leur temps à l'examen de ce travail,

M[r] Vincent CORDONNIER, Professeur à l'Université des Sciences et Techniques de Lille-Flandres-Artois, qui a su donner un avis autorisé sur ces travaux,

Et bien sûr, M[r] Michel MERIAUX, chargé de recherche au CNRS, qui m'a proposé ce sujet, et m'a judicieusement conseillé tout au long de l'évolution de ce travail.

Je souligne aussi l'aide apportée par les membres de l'équipe graphique avec lesquels j'ai eu de nombreuses discussions fructueuses.

J'ai apprécié l'ambiance agréable entretenue par les membres du LIFL et les techniciens du laboratoire, et je remercie M[r] Henri GLANC qui a reproduit avec diligence ce document.

Enfin, je n'omettrai pas de signaler que sans le soutien de ma famille je n'aurai pu accomplir sereinement ce travail.

PLAN

PLAN

INTRODUCTION

0. INTRODUCTION

La diminution du coût de revient des micro-processeurs et les possibilités croissantes d'intégration, grâce notamment aux techniques V.L.S.I. (Very Large Scale Integration), ont conduit de nombreux laboratoires de recherche à développer des architectures multiprocesseurs.

Depuis l'avènement de la technique dite W.S.I. (Wafer Scale Integration) les perspectives sont encore plus vastes. L'intégration de réseaux cellulaires dans le silicium est désormais envisageable sérieusement. Quelques machines ont d'ailleurs déjà été réalisées.

Ces machines parallèles sont particulièrement appropriées aux applications réclamant des performances de calcul importantes; leur emploi pour le traitement graphique était donc tout indiqué.

L'image de synthèse est de plus en plus utilisée dans les logiciels actuels, quels que soient leurs domaines d'applications. Outre ses qualités artistiques intrinsèques, l'image est un moyen particulièrement performant de véhiculer un grand nombre d'informations. La perception visuelle par l'opérateur humain est très rapide, mais pour que le dispositif soit cohérent, l'affichage de l'image doit avoir des performances en rapport.

La synthèse d'images requiert des temps de traitement prohibitifs eu égard à l'interactivité et au temps réel. Par exemple, l'élaboration d'une image composée de 512x512 pixels (abréviation pour picture element) réclame, à raison d'un traitement élémentaire de 100 µs par point, plus de 26 secondes. Or, à une image haute définition correspond de nos jours une résolution de l'ordre de 4096x4096 pixels. C'est là la justification principale des machines multiprocesseurs dédiées à la synthèse d'images.

Il y a deux façons d'appréhender le problème de l'accroissement des performances :

- Soit par la structuration de l'image en une scène composée de diverses entités élémentaires. Cela aboutit à une distribution des objets sur les différents processeurs : Machines Objets.
- Soit par l'amélioration de la visualisation elle même. L'image est alors découpée en sous-espaces, chacun étant confié à un processeur. Dans le cas particulier des Machines Pixels, à chaque pixel est associé une cellule (processeur élémentaire).

C'est à ces dernières que nous nous intéressons dans ce travail.

La conception des machines de ce type a, dans la plupart des cas, suivi le processus classique : réalisation du réseau cellulaire physique proprement dit, suivi du développement de logiciels adaptés à ces architectures, les algorithmes existants étant caducs car conçus pour des machines de type VON NEUMANN.

Notre approche est quelque peu différente puisqu'il s'agit de déduire à partir de l'étude de la parallélisation des algorithmes de base de la synthèse d'images, l'architecture cellulaire la mieux adaptée.

Le chapitre I fixe le cadre architectural dans lequel nous évoluerons. Il n'est pas raisonnable d'étudier tous les types d'architectures, le choix est guidé par l'adéquation avec le problème traité.

Le chapitre II traite de l'algorithme le plus fréquemment utilisé en synthèse d'images : le tracé de segments. Ce travail permet dès lors la définition de plusieurs architectures potentiellement intéressantes.

Le chapitre III aborde le problème du tracé de cercles. De nouvelles contraintes apparaissent, mais les architectures sont sensiblement identiques.

Le chapitre IV s'attaque à un problème plus complexe, celui du remplissage. Cette étude nous impose un éventail d'architectures plus strict.

Le chapitre V enfin, dresse un inventaire des différentes architectures envisagées, en détaillant plus avant celles offrant un intérêt particulier. Les performances obtenues sont comparées et les perspectives d'évolutions, en vue de résoudre d'autres problèmes, sont évaluées. C'est alors qu'intervient le choix final.

Une simulation destinée, entre autres, à valider tant les algorithmes que les architectures proposés, fait l'objet d'un travail entrepris par A. ATAMENIA.

Les résultats déjà obtenus, nous permettent d'envisager sérieusement un développement ultérieur.

LE CONTEXTE ARCHITECTURAL

TABLE DES MATIERES

1. LE CONTEXTE ARCHITECTURAL

Les algorithmes parallélisés développés dans ce travail doivent nous permettre de définir quelques architectures spécialisées dans la synthèse d'images. Cependant pour concevoir ces algorithmes, il faut se fixer dès à présent un cadre architectural minimum.

Dans les pages qui suivent nous préciserons quelles sont les restrictions que nous adoptons pour notre étude.

Mais avant cela, précisons un point qui n'est pas une contrainte, mais une particularité de notre réseau : à chaque cellule est associé un pixel de l'écran. En effet, notre objectif est la conception de ce que l'on appelle communément une machine pixel.

Remarque :

Si les possibilités techniques du moment ne permettent pas la réalisation d'un réseau de cellules suffisamment étendu, l'image est traitée morceau par morceau. Les manières, plus ou moins complexes, de subdiviser l'image ont fait l'objet de nombreuses études [ARO 86], [ARO 87], [POT 83], [ROS 83]... L'hypothèse de travail que nous faisons est qu'il est possible d'utiliser des cellules suffisamment simples pour pouvoir réaliser physiquement un réseau 1000X1000.

1.1. HISTORIQUE

[COD 68] [NEU 66] [PRD 84] [ROS 83] [WOL 86]

Les bases d'un automate cellulaire auto-reproducteur furent énoncées dès les années 50 par J. VON NEUMANN lui-même. Ses travaux furent menés à terme par A.W. BURKS [NEU 66]. Les automates cellulaires bidimensionnels furent étudiés pour la première fois en tant que calculateurs par J.F. CODD [COD 68]. En 70 J.H. CONWAY rendit célèbre les automates en présentant son modèle du "Jeu de la vie".

Enfin, en 1980 S. WOLFRAM entrepris une classification des différents automates cellulaires étudiés qui donna lieu à un ouvrage assez exhaustif sur "La théorie et les applications des automates cellulaires" [WOL 86].

Les réseaux cellulaires que nous étudions, bien qu'issus de ces réflexions sont quelque peu différents, en ce sens qu'il ne s'agit plus réellement d'un réseau d'automates, mais plutôt d'un réseau de mini-calculateurs [PRD 84], [ROS 83].

1.2. QU'EST CE QU'UNE MACHINE CELLULAIRE ?

Un réseau cellulaire est un ensemble de cellules, chacune d'entre elles étant considérée comme un processeur élémentaire.

Un réseau cellulaire est caractérisé par [HEM 86] :

- la topologie,
- la connectique,
- la taille,
- le mode de synchronisation,
- la communication avec l'extérieur,
- la nature des cellules.

1.2.1. LA TOPOLOGIE

C'est la manière dont est agencé le réseau, c'est à dire la position relative des cellules. Diverses solutions sont envisagées mais la régularité du maillage est une constante importante facilitant la gestion de l'ensemble.

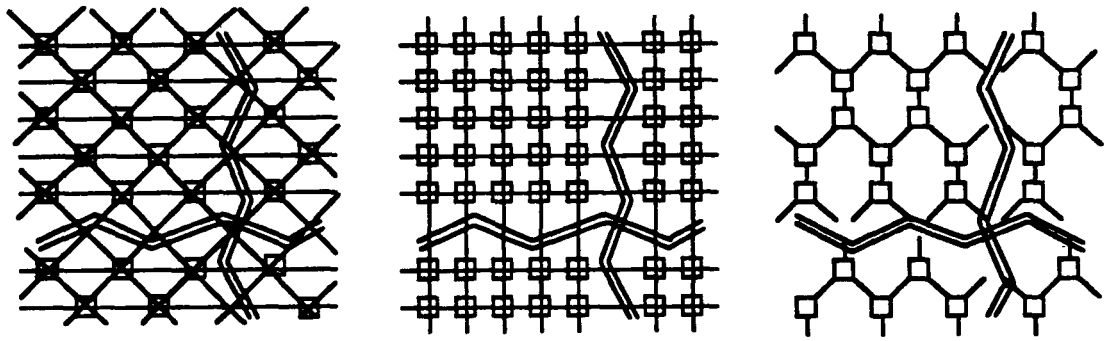


Figure 1 : Différents maillages

Intuitivement, la topologie carrée est mieux adaptée aux problèmes d'imagerie, ceci étant dû évidemment à l'association directe pixel-cellule. Mais le positionnement physique n'a que peu d'importance s'il n'est pas directement associé aux liaisons inter-cellulaires.

En fait, nous allons choisir la topologie carrée avec une connectique simple, car c'est celle qui nous semble la mieux adaptée au problème à résoudre.

1.2.2. LA CONNECTIQUE

Plus que la topologie, ce sont les liaisons inter-cellulaires qui différencient les réseaux et permettent de les classifier.

1.2.2.1. RESEAU TOTALEMENT CONNECTE

C'est le cas des réseaux neuronaux notamment. Chaque cellule est connectée à toutes ses congénères. Le nombre de liens nécessaires est en $O(N^2)$ où N est le nombre de cellules.

De nouvelles techniques permettent d'envisager sérieusement cette éventualité, en particulier les solutions optiques. A l'aide de simples lentilles, ou d'un hologramme moins gourmand en énergie lumineuse, et d'un laser il est possible de réaliser des milliers d'interconnexions sans interférences. En outre, les communications se font à la vitesse de la lumière [GHL 88], [RGZ 87].

1.2.2.2. RESEAU A LIAISONS LOCALES

Une cellule n'est connectée qu'à ses voisines directes.

Exemples :

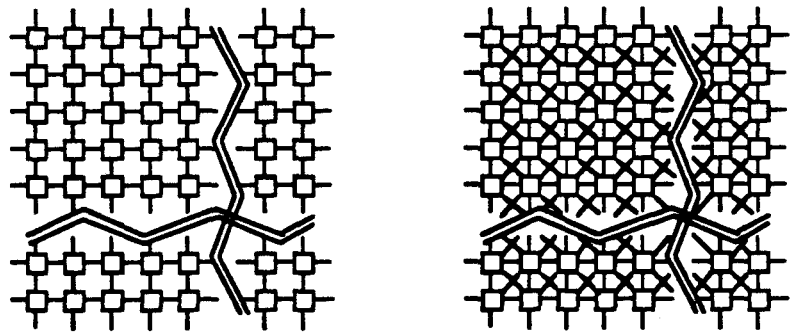


Figure 2 : Réseaux carrés à 4 et 8 voisins

Ce sont les plus classiques.

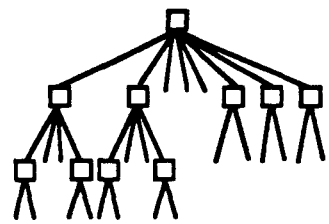


Figure 3 : Réseau arborescent

Remarque :

Sur certaines architectures quelques chemins entre frères du même niveau sont parfois ajoutés pour gagner du temps lors des communications.

Cas particulier : Arbre quaternaire

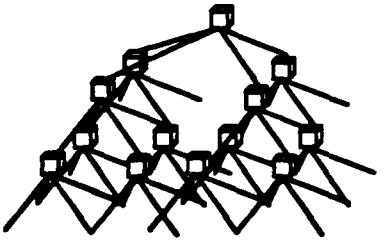


Figure 4 : Réseau pyramidal

C'est la forme arborescente la plus employée en imagerie.

Remarque :
Les architectures arborescentes renferment une notion de hiérarchie qui ne se trouve pas dans le réseau carré.

L'architecture pyramidale présente de nombreux atouts dont le principal est l'accès rapide à une cellule quelconque du réseau.

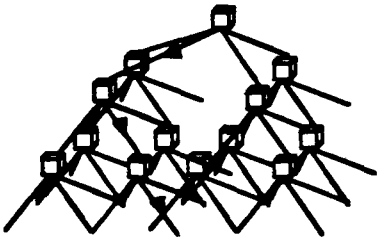


Figure 5 : Accès dans un réseau pyramidal

Pour un écran de définition NXN le temps d'accès aux cellules est en $O(\log_4(N^2))$ avec une architecture pyramidale. Alors que pour un réseau carré classique à accès par un bord, le temps d'accès moyen est proportionnel à N/2.

Pour utiliser de façon optimale une telle architecture, il faut concevoir des algorithmes manipulant des codes, et non plus les coordonnées classiques pour repérer les points. Ces codes sont obtenus par des divisions récursives (par 4 ou 8) de l'image. Celle-ci est conservée sous la forme d'un arbre quaternaire (quadtree) ou octal (octree) [ABJ 85], [SHS 87]. La géométrie classique n'y est transposable qu'au prix de nombreuses modifications [GAL 87], [PEL 85].

Des modèles de parallèlisation existent déjà pour ces machines [BHR 88].

Entre les réseaux totalement connectés et la stricte liaison locale, il est possible de définir des solutions intermédiaires conservant des liaisons distantes. Par exemple, sur le réseau réduit du cosmic cube qui ne compte que 64 noeuds, une architecture de type hypercube est adoptée [SEI 85].

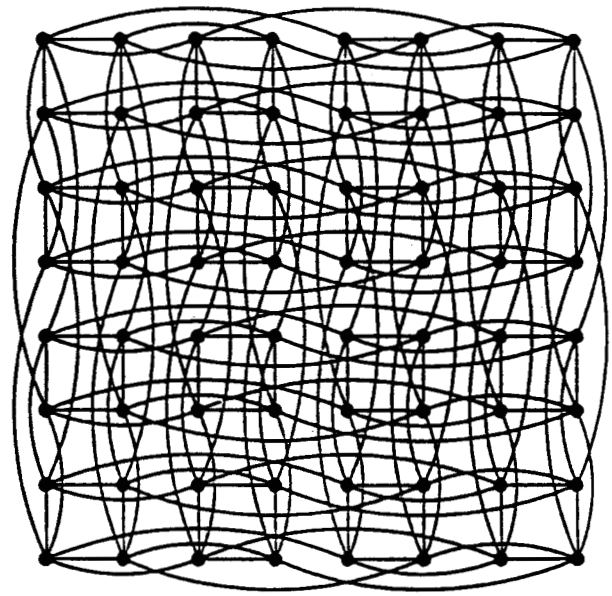


Figure 6 : Projection bidimensionnelle d'un hypercube de dimension 6

La gestion des communications dans un tel réseau est sensiblement plus complexe.

En synthèse d'images les solutions les plus utilisées sont les architectures carrées à quatre ou huit voisins et l'architecture arborescente. Les deux approches sont assez différentes et intéressantes pour justifier, pour chacune d'entre elles, une étude complète.

Nous écartons de notre travail les architectures arborescentes et de type hypercube, pour nous consacrer essentiellement aux réseaux carrés qui nous semble mieux adaptés.

1.2.2.3. RESEAU NON CONNECTE

Aucune liaison physique n'existe entre les cellules. Son utilisation est limitée aux solutions S.I.M.D., pour lesquelles chaque cellule exécute sa tâche sans se préoccuper de ses voisines.

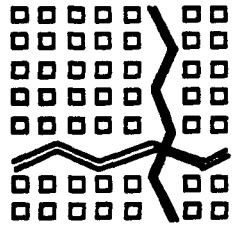


Figure 7 : Réseau carré non connecté

En définitive, nous avons choisi de limiter notre étude aux réseaux carrés, mieux adaptés à la géométrie et en corrélation directe avec l'image, sans préjuger de leur connectique (i.e. totalement, partiellement ou non connectés).

1.2.3. LA TAILLE DU RESEAU

Il s'agit du nombre de cellules qui composent le réseau. Ce nombre est en relation directe avec la définition de l'écran, puisqu'il s'agit, rappelons le, de machines pixels.
La première contrainte est la possibilité d'implantation sur une pastille de silicium. D'autres paramètres tels que les possibilités de subdivision, en vue d'une implantation sur plusieurs chips soulevant des problèmes de raccordement, sont également à prendre en compte.

Pour un écran 512X512 il faut 256000 cellules. Ce chiffre auparavant rédhibitoire, ne l'est plus désormais grâce aux nouvelles techniques d'intégration. Notons qu'il existe d'ores et déjà une machine massivement parallèle dédiée à l'intelligence artificielle : la "connection machine" [HIL 88], disposant de 65000 cellules fortement connectées. Outre les cellules, un réseau de 4000 routeurs permet de relier entre eux des blocs de 16 cellules connectées.

1.2.4. LE MODE DE SYNCHRONISATION

Il faut distinguer la synchronisation physique du réseau (horloge commune à toutes les cellules), de la synchronisation du traitement. Cette dernière étant nécessaire au bon fonctionnement de certains algorithmes pour lesquels les cellules ont besoin d'informations sur le traitement effectué par leurs voisines.

1.2.4.1. LE CONTROLE PHYSIQUE

RESAU SYNCHRONE :

Les cellules sont pilotées par un même front d'horloge, ce qui pose un problème de câblage non trivial dans le cas de réseaux étendus, comme ceux que nous étudions. Le principal attrait de cette solution réside dans sa simplicité de gestion, et le déterminisme total de son fonctionnement.

RESAU ASYNCHRONE :

Chaque cellule possède sa propre horloge. Pour qu'elle puisse communiquer avec une de ses voisines, il faut mettre en place un protocole de communication, ce qui évidemment nuit aux performances.

1.2.4.2. LE PILOTAGE FONCTIONNEL

LE MODE SYNCHRONE :

Le mode synchrone se caractérise par un cycle commun à tout le réseau.

Ce cycle se décompose en trois étapes :

- Acquisition de l'information : l'information est, dans la pratique, la commande suivie de la liste de ses paramètres.
- Traitement de l'information : interprétation de la commande.
- Exécution effective de la commande sur les données internes et transmises. Puis, éventuellement transfert à destination de la (ou des) cellule(s) suivante(s).

Remarques :

- Il y a cadencement de la transmission des données, chaque cellule fournit les informations à la cellule suivante au rythme de la commande la plus lente.
- Ce mode de pilotage induit un fonctionnement rigide et une algorithmique particulière. Il est généralement employé pour les réseaux systoliques [FRI 83], [KUN 87].

LE MODE ASYNCHRONE :

Chaque cellule du réseau travaille indépendamment de ses voisines :

- le cycle d'une cellule est différent de ceux des cellules voisines,
- chaque cellule exécute son propre travail sans se soucier de ses voisines.

=> Les mécanismes de communication sont plus complexes et plus lourds à mettre en place.

Il est nécessaire d'établir un protocole de communication, capable de résoudre les problèmes suivants :

- Permettre la communication entre cellules voisines
  Tester si la cellule voisine est libre
  Synchroniser les cellules pour permettre l'échange

- Eviter les inter-blocages

- Arbitrer les demandes simultanées d'accès à une cellule

- Minimiser le temps de communication

Le Cosmic Cube [SEI 85], déjà vu plus haut, est un réseau asynchrone tout comme les machines à front d'ondes [KUN 82].

Remarques :

- La mise en oeuvre d'un protocole de communication permet à l'utilisateur de mettre en place différentes stratégies sans avoir à chaque fois à redéfinir complétement son réseau physique de cellules. Mais par ailleurs, l'échange d'informations n'étant pas systématique, le temps que la cellule passe à gérer ses communications peut devenir non négligeable, et donc pénaliser les performances du réseau.

- Le mode asynchrone autorise une grande disparité au niveau des cellules, ce qui n'était pas possible avec le mode synchrone.

- Le nombre de cycles ainsi que la durée d'exécution d'une commande par une cellule n'influent pas sur le comportement du reste du réseau, si ce n'est au niveau des performances.

Le mode de pilotage adopté ne préjuge pas totalement du fonctionnement du réseau. Reprenons la classification de FLYNN :

Une machine S.I.M.D. (Single Instruction, Multiple Data) peut être synchrone ou asynchrone, avec néanmoins une préférence pour le synchrone puisque toutes les cellules effectuent le même travail.

Pour un fonctionnement M.I.M.D. (Multiple Instruction, Multiple Data) le mode asynchrone facilite l'exploitation du parallélisme. Néanmoins, certaines solutions particulières sont contrôlées de façon synchrone.

1.2.5. LA COMMUNICATION AVEC L' EXTERIEUR

Une machine cellulaire pour l'imagerie est constituée en pratique de trois éléments distincts :

- la machine hôte,
- le réseau de cellules,
- l'écran.

Les communications sont donc de deux ordres :

- en entrée : hôte -> réseau
- en sortie : réseau -> écran.

1.2.5.1. LE TRANSFERT HOTE->RESEAU

Nous connaissons le rôle du réseau cellulaire, définissons celui de l'ordinateur hôte.

C'est par son intermédiaire que l'utilisateur transmet ses ordres au réseau. Il joue le rôle d'interface homme-réseau en préparant les commandes et en les injectant dans le réseau. En phase de développement, il permet également de charger les micro-programmes dans les cellules. Enfin, il contrôle la bonne exécution des commandes qu'il émet.

La partie préparation des commandes, principalement le calcul de leurs paramètres, est plus ou moins complexe, elle est en fait, inversement proportionnelle à la complexité du micro-programme cellulaire correspondant.

Plusieurs options sont envisageables pour communiquer la commande paramétrée au réseau.

Voici une liste représentative des choix possibles pour un réseau carré :

1.2.5.1.1. DISTRIBUTION TOTALE

La même commande est distribuée dans tout le réseau simultanément.

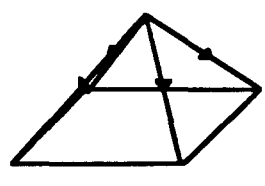


Figure 8 : Distribution totale

Cette diffusion simultanée, encore illusoire il y a peu de temps, trouve une réponse dans les machines optiques où les cellules munies de photo-capteurs reçoivent la commande en parallèle [PAB 84]. Des solutions à bus optiques existent également [CML 87].

Cas particulier :

La commande est transmise à une partie du réseau seulement, mettant ainsi à profit le fait que, souvent, seules quelques cellules sont concernées par une même commande.

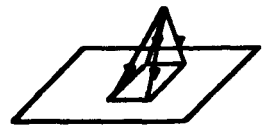


Figure 9 : Distribution partielle

Cette solution nécessite elle aussi l'apport des techniques optiques complexes.

Plus classiquement on s'oriente généralement vers une des options suivantes :

1.2.5.1.2. ENVOI A UNE CELLULE UNIQUE

Chaque cellule se charge de transmettre la commande à ses voisines, elles-mêmes la transmettant à leurs voisines etc...

Ce comportement est communément appelé propagation.

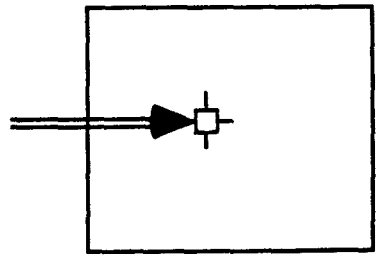


Figure 10 : Accès unique

Cette solution est en fait double :

- soit la cellule initiale est fixe et il s'agit d'une solution câblée,
- soit la cellule initiale est fonction des données et il faut alors disposer d'un routage pour l'atteindre. Ce routage peut être soit logiciel, et les performances s'en ressentent, soit physique et il agit en tâche de fond par rapport à l'application [MAZ 88], c'est-à-dire en parallèle géré par l'architecture elle-même.

1.2.5.1.3. ACCES AU RESEAU PAR UN BORD

Une commande unique transmise à tout le bord du réseau :

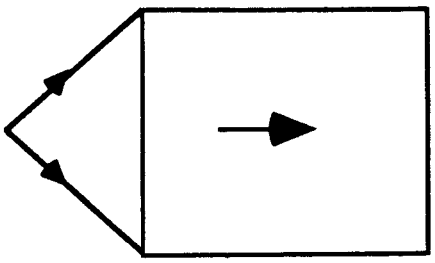


Figure 11 : Accès par un bord entier

Comme pour la diffusion, la commande peut n'être transmise qu'à une partie du réseau, ce qui permet d'envisager un envoi parallèle des commandes.

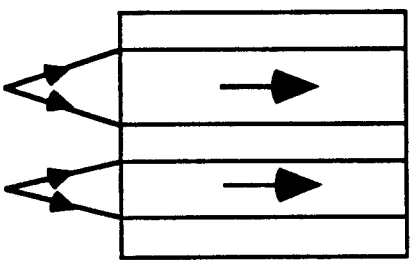


Figure 12 : Accès sélectif sur un bord

Remarque :

- Il ne faut pas confondre la manière dont sont diffusées les commandes avec le fonctionnement de l'algorithme. Autrement dit, il faut distinguer le transport des commandes de leur traitement.

  Ainsi, une machine S.I.M.D. se satisfait avec plus ou moins de bonheur de ces diverses situations. Mais, plus généralement, la manière dont sont véhiculées les commandes implique un type de propagation optimale. La diffusion à partir d'une cellule unique, par exemple, induit un comportement de type propagation, et l'accès par un bord, un fonctionnement pipeline...

- Il existe une corrélation évidente entre le mode d'accès et les liens physiques du réseau. Par exemple, une distribution totale est essentiellement destinée à un réseau non connecté.

1.2.5.2. LE TRANSFERT RESEAU->ECRAN

Le problème est la manière dont on va extraire les informations contenues dans le réseau pour la visualisation.

i) La solution la plus immédiate est que la cellule gère directement le pixel qui lui est associé. Il s'agit là encore d'une technique d'avant-garde nécessitant une forte intégration.

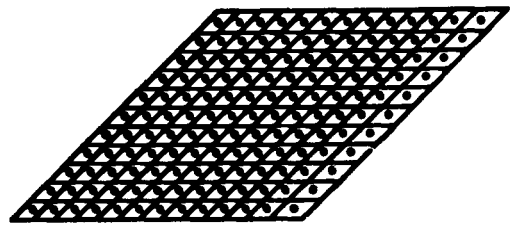


Figure 13 : Ecran constitué de cellules

Des écrans à cristaux liquides suivant ce mode sont d'ores et déjà étudiés par le CNET [FOU 88], en vue de réalisation du visiophone.

ii) Autre solution : la mémoire de trame à double accès utilisée par les cellules pour ranger leurs informations, et par l'affichage pour le rafraîchissement. Elle requiert un grand nombre de connexions ou des bus très rapides [POT 83].

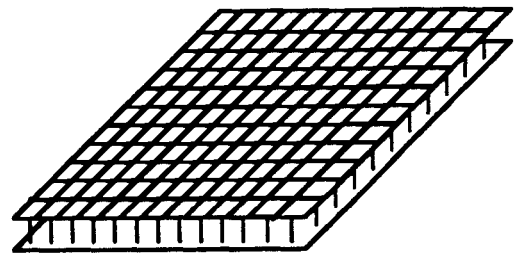


Figure 14 : Cellules connectées à la mémoire de trame

iii) La solution du registre à décalage est la plus classique, mais pose des problèmes de synchronisation.

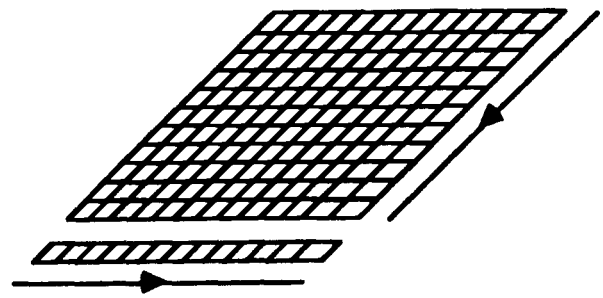


Figure 15 : Registres à décalages

Les informations transitent dans le réseau de manière synchrone et sont récupérées dans un registre à décalage pour l'affichage. Cela est réalisé en tâche de fond indépendamment du travail effectué dans les cellules.

1.2.6. LA NATURE DES CELLULES

Deux approches sont envisageables :

- Réseau homogène :
  Toutes les cellules du réseau sont identiques, ce qui entraîne une grande simplicité architecturale et des communications simplifiées.

- Réseau hétérogène :
  Les cellules entrant dans la composition du réseau ne sont pas toutes identiques. Cela peut être probant malgré la complexité qui en découle. Dans ce cas le réseau est nécessairement controlé de manière asynchrone.

Les différentes caractéristiques des cellules du réseau entrent aussi en jeu dans la définition de sa composition :

- mémoire disponible pour chaque cellule
- nombre de registres de travail
- les diverses opérations possibles
  etc...

La gamme s'étend depuis une cellule logique combinatoire jusqu'à un processeur complet.

1.3. LES MACHINES EXISTANTES

Un certain nombre de machines d'imagerie, composées de nombreux processeurs éléments organisées en réseaux, ont déjà vu le jour [HFU 83], [KID 83], [PIE 88].

Nous en présentons quelques unes pour situer les techniques déjà opérationnelles.

1.3.1. PIXEL-PLANES

La plus connue des machines pixels actuellement développée est Pixel-Planes de FUCHS [FUC 81]. Pour cette machine les problèmes de synthèse d'images sont ramenés à la résolution de formes linéaires du type $Ax + By + C$.

Le schéma fonctionnel est le suivant : les multiplications sont effectuées sur deux arbres binaires d'additioneurs, tandis que le réseau est constitué de cellules réduites également à de simples additioneurs [FUC 85].

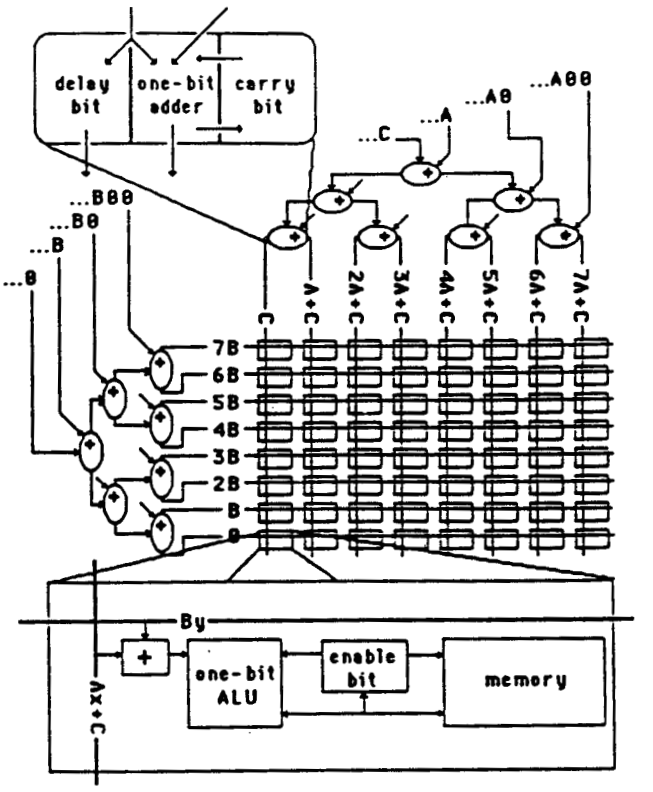


Figure 16 : Pixel-Planes 1985

Pour Pixel-Planes4 le nombre d'additionneurs passe à N+1 (pour un réseau NXN), afin de fournir à chaque cellule Ax + By + C directement [FUC 88].

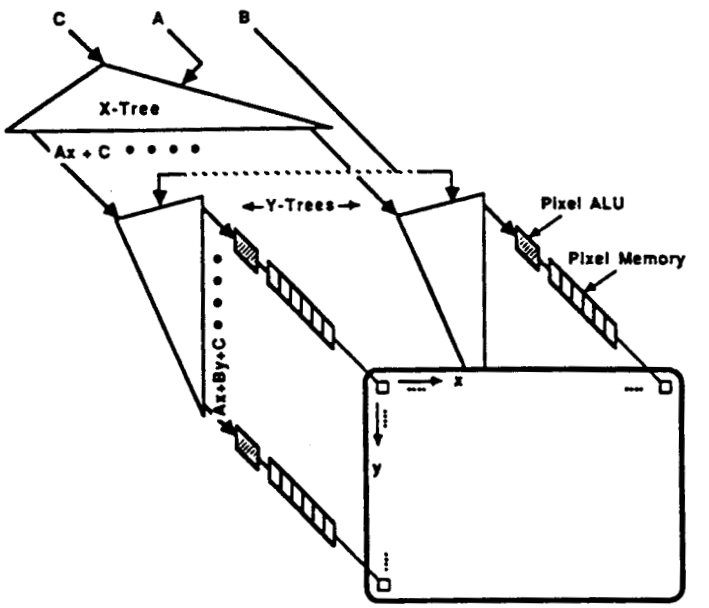


Figure 17 : Pixel-Planes 1988

Enfin, le projet Pixel-Planes5 ou Pixel-Power [FUC 88], apporte, outre des améliorations technologiques (augmentation de la vitesse d'horloge, réduction de l'encombrement, etc...), des modifications fonctionnelles. Désormais, la machine est fondée sur la résolution d'expressions quadratiques de la forme $Dx^2 + Exy + Fy^2 + Ax + By + C$. Cela permet notamment d'appliquer des modèles d'ombrages tels que celui de PHONG [PHO 75], ou encore d'avoir des calculs de sphères plus rapides.

Remarques :

- Au niveau du réseau carré lui-même on peut considérer le fonctionnement comme étant de type SIMD. La majeure partie du travail est effectuée soit par l'ordinateur hôte, pour la préparation des paramètres de l'équation, soit par les arbres de distribution, pour l'évaluation de l'équation. La tâche des cellules se limite à quelques opérations arithmétiques ou logiques locales aux pixels.

- De plus, il s'agit d'un traitement sérialisé (effectué bit par bit) ce qui conduit à des entités physiques extrêmement simples. En revanche, elles sont particulièrement nombreuses : outre le réseau de NXN cellules, il y a N+1 arbres constitués chacun de N-1 additionneurs. Le principal avantage réside dans le temps d'accès à une cellule qui est en O(log(N)), correspondant aux transits successifs dans les arbres de profondeur log(N).

1.3.2. MPP

[BAT 80] [POT 83]

Le "Massively Parallel Processor" n'est pas destiné à la synthèse mais plutôt au traitement d'images mais son architecture se rapproche de celles que nous allons étudier. Il s'agit en effet d'un réseau de 128X128 processeurs éléments controlés de façon S.I.M.D..

Les composantes principales en sont :
- le contrôleur séquentiel,
- le réseau de processeurs,
- la mémoire intermédiaire.

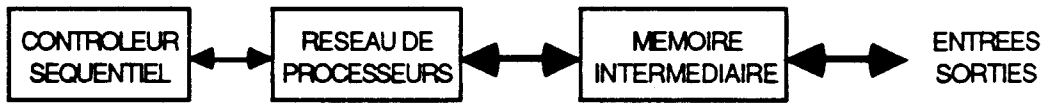


Figure 18 : Organisation de MPP

Chaque cellule a accès à une partie de la mémoire intermédiaire contenant, entre autres, l'image à traiter.

Le réseau n'étant pas assez étendu, l'image est traitée morceau par morceau, d'où l'utilité du bus à 320 Mbytes par seconde pour le double accès à la mémoire intermédiaire.

Le réseau est utilisé pour traiter des parties d'images, le découpage adopté est des plus simples :

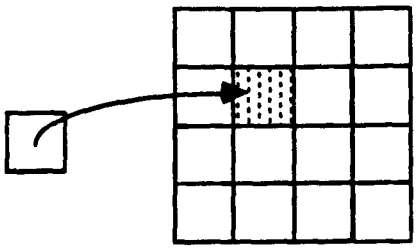


Figure 19 : Affectations successives du processeur

Nous ne pouvons pas comparer cette machine à nos propositions, puisqu'il s'agit rappelons le d'une machine destinée au traitement d'images, elle sert simplement de référence technique.

1.4. LES MACHINES ETUDIEES

Nous avons vu diverses façons d'accéder au réseau. Chacune d'entre elles engendre une architecture particulière tenant compte du mode de contrôle et de commande.

Nous énumérons dès à présent, pour plus de clarté, les différentes architectures auquelles nous avons abouties lors de l'étude de la parallélisation des algorithmes. Pour cette présentation nous nous référons au cas du tracé de segments.

1.4.1. PRESENTATION

1.4.1.1. DISTRIBUTION TOTALE

Avec cette appproche, le travail de l'ordinateur hôte consiste simplement à transmettre à chacune des cellules la même commande dont il a évalué les paramètres.

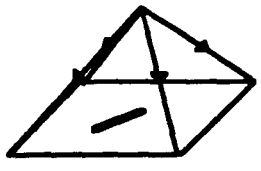


Figure 20 : Distribution de la commande

Ce premier mode de fonctionnement du réseau envisagé est appelé S.I.M.D. (Instruction Simple, Données Multiples).

La façon dont est diffusée la commande suppose un développement architectural à ne pas négliger.

Il y a plusieurs manières d'aborder ce problème :

- Soit en respectant la simultanéité de l'émission, auquel cas un câblage classique est désuet. L'apport des techniques optiques est indispensable. Les commandes sont transmises à toutes les cellules (munies de photo-capteurs) en une seule étape, au moyen d'un laser et d'un hologramme, ou plus simplement de lentilles.
  Dans cette hypothèse le réseau est contrôlé de façon tout à fait synchrone.

- Soit en dissociant l'émission des commandes de leur traitement. Toutes les solutions peuvent être envisagées pour la diffusion, multipipeline ou propagation pure notamment.

  L'exécution des commandes est :
  - soit totalement synchrone (dans tout le réseau) ce qui suppose que l'exécution de la commande est différée après la diffusion complète,
  - soit partiellement synchrone (sur une colonne pour une diffusion multilinéaire),
  - soit asynchrone, exécution de la commande dès réception, indépendamment des cellules voisines.

  L'hypothèse la plus simple est celle d'un contrôle synchrone du réseau. Dans les autres cas, il est possible d'utiliser le mode de diffusion pour allèger la tâche des cellules, c'est d'ailleurs ce qui est proposé pour les algorithmes dérivés de l'évaluation de l'équation.

Notons que la synchronisation d'un réseau 1000X1000 ne va pas sans poser quelques problèmes architecturaux.

Faisons abstraction du mode de diffusion qui est étroitement lié à la réalisation physique du réseau.

Deux approches sont possibles :
- purement SIMD, le tracé d'un segment nécessite la diffusion successive de toutes les instructions du programme. Le contrôleur est centralisé.
- SPMD (Simple Program, Multiple Data), une seule commande est envoyée, activant une suite d'instruction. Le contrôleur est distribué.

D'autre part, la synchronisation étant totale, le problème de l'arrêt ne pose aucune difficulté :
- soit on évalue a priori la durée d'exécution
- soit on espionne l'activité des cellules

1.4.1.2. ACCES PAR UN BORD

I) MULTIPIPELINE SIMPLE

Dans ce réseau nous considérons que l'information se déplace unilatéralement de la gauche vers la droite.

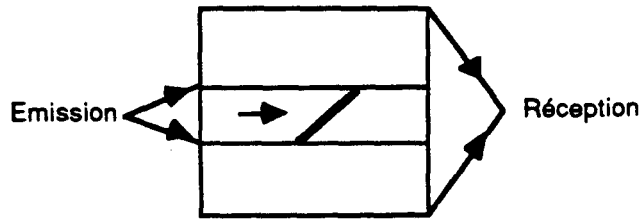


Figure 21 : Multipipeline simple

Les commandes sont récupérées sur le bord opposé au bord d'émission, afin de contrôler la totale exécution de la commande. Ce fonctionnement est appelé MULTIPIPELINE SIMPLE.

Sur ce type de réseau, avec ce genre de propagation, il est possible de réduire encore le travail des cellules en s'intéressant à l'accès au réseau.

II) MULTIPIPELINE A VECTEUR D'ENTREES

Au lieu de transmettre une même information à toutes les cellules du réseau, chaque cellule reçoit la commande avec des paramètres spécifiques.

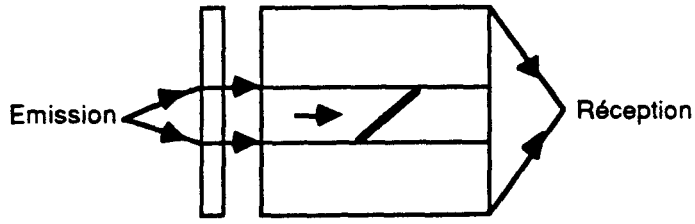


Figure 22 : Multipipeline à vecteur d'entrées

La valeur d'entrée est remplacée par un vecteur d'entrées.

Cette nouvelle solution est simplement appelée MULTIPIPELINE A VECTEUR D'ENTREES. Il ne s'agit d'ailleurs pas d'un nouveau fonctionnement, mais plutôt d'une option architecturale.

Nous ne discutons pas ici de la réalisation du vecteur d'entrées, nous considérons la solution multipipeline dans son ensemble.

Le réseau est unidirectionnel : toutes les communications inter-cellulaires se font de la gauche vers la droite ce qui simplifie la gestion des communications.

Le problème du test d'arrêt est résolu par la récupération des commandes sur le bord du réseau opposé au bord d'émission.

Chaque cellule doit exécuter la même commande avec des paramètres différents et les cellules d'une même colonne exécutent la même instruction. Le contrôle du réseau peut donc être synchrone, de la même manière qu'un réseau systolique.
Un pilotage asynchrone augmente le parallélisme mais complexifie le contrôle de l'ensemble.

Le multipipeline est une solution architecturale peu coûteuse à réaliser et disposant d'un parallélisme important.

Mais, le nombre de liaisons inter-cellulaires étant pour le moins limité, il faudra vérifier que, quel que soit le problème posé en synthèse d'images, il existe une solution algorithmique performante sur cette architecture.

1.4.1.3. ACCES PAR UNE CELLULE

Le mode de fonctionnement associé à ce type d'accès est la propagation cellulaire.

Trois approches sont envisageables :

i) LA PROPAGATION PURE

La propagation pure consiste à diffuser, à partir de la cellule centrale, la commande dans tout le réseau.

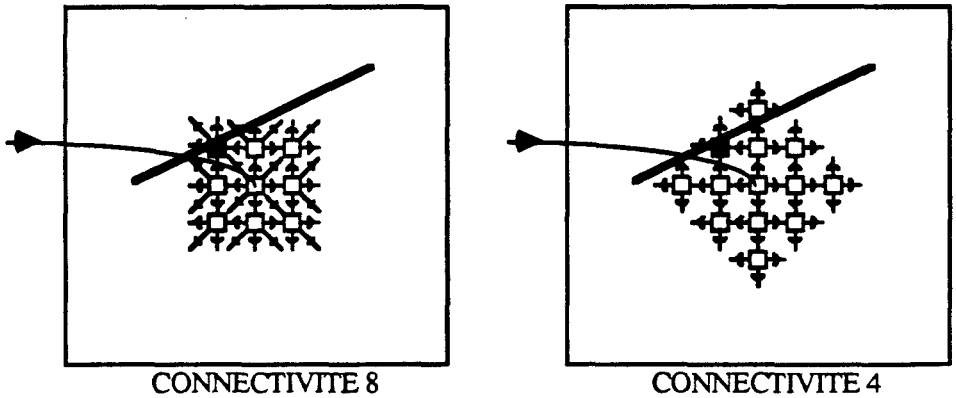


Figure 23 : Propagation pure

Chaque cellule transmet la commande avant d'effectuer le traitement.

REMARQUES SUR LA PROPAGATION :

La propagation simple dans les huit directions n'est pas performante de par le fait que :

- bon nombre des transmissions sont inutiles, elles sont destinées à des cellules ayant déjà effectué le traitement,
- de nombreux conflits sont engendrés.

Même en supposant qu'une cellule reconnait les cellules qui lui ont transmis la commande et ne leur renvoie pas le message, les conflits d'accès restent nombreux. En fait, les communications ne sont plus réellement inutiles puisqu'elles limitent le nombre de directions de propagation au tour suivant.

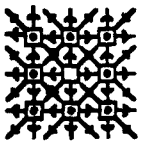


Figure 24 : Conflits sur un réseau à 8 voisins

Une cellule activée une première fois, peut recevoir jusqu'à quatre fois la même commande lors du cycle suivant.

Une cellule libre quant à elle, reçoit jusqu'à trois exemplaires du même message.

En réduisant la transmission à quatre voisines, moins les cellules émettrices, les communications inutiles et les conflits d'accès sont certes moins nombreux, mais ils subsistent néanmoins :

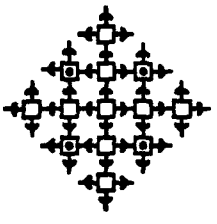


Figure 25 : Conflits sur un réseau à 4 voisins

Nous proposons ci-dessous un mode de propagation permettant d'éviter les conflits d'accès, chaque cellule ne reçoit qu'une fois le message :

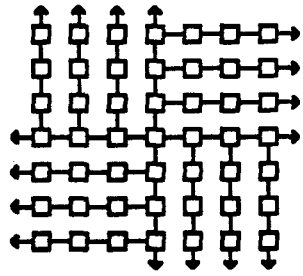


Figure 26 : Propagation sans conflits (4 voisins)

La loi de propagation s'énonce comme suit :

- Toutes les cellules transmettent le message à la cellule située à l'opposé de la cellule émettrice.

Figure 27 : Transmission unidirectionnelle

- Les cellules situées sur la même ligne ou la même colonne que la cellule initiale, effectuent, en plus de la transmission à l'opposé, un envoi dans la direction perpendiculaire à celle-ci, à droite ou à gauche mais dans le même sens pour toutes les cellules.

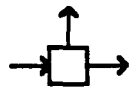


Figure 28 : Transmission bidirectionnelle

Au niveau des cellules, sans considérer comment l'information est transmise, le comportement de cette propagation contrôlée est identique à celui de la transmission simple à quatre voisines.

Remarque :

Le temps d'accès à une cellule est jusqu'à deux fois plus long pour une propagation à quatre voisins pour les cellules situées sur les diagonales, par rapport à une méthode à huit voisins.

D'où l'intérêt de développer une propagation contrôlée pour un réseau à huit voisins.

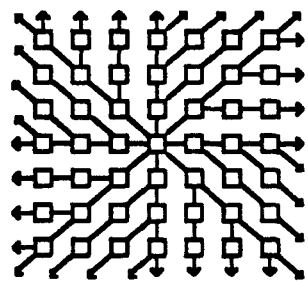


Figure 29 : Propagation sans conflits (8 voisins)

Le comportement est fortement inspiré de celui pour quatre voisins, si ce n'est qu'il y quatre axes sur lesquels la propagation est bidirectionnellle.

L'énoncé de la loi de propagation est donc le suivant :

- toutes les cellules transmettent le message à la cellule opposée à la cellule émettrice :

Figure 30 : Transmission unidirectionnelle

- les cellules situées sur un des quatre axes, effectuent en plus un envoi dans la direction située à droite de celle-ci.

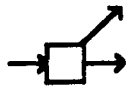


Figure 31 : Transmission bidirectionnelle

Comment implémenter ces modes de propagation :

- soit en incorporant en tête de message les coordonnées de la cellule initiale,
- soit en incluant dans ledit message une valeur indiquant s'il s'agit ou non d'une transmission unidirectionnelle,
- soit par un câblage approprié.

La solution câblée est la plus performante.

Le transfert d'informations entre l'ordinateur hôte et la cellule centrale est également câblé.

La cellule centrale est physiquement différente des autres, un lien supplémentaire est directement relié à l'ordinateur hôte. C'est une contrainte à ne pas négliger pour l'intégration dans le silicium, de même que le maillage régulier par parties.

Le test d'arrêt s'effectue comme pour les solutions multipipelines par récupération des commandes sur les bords du réseau.

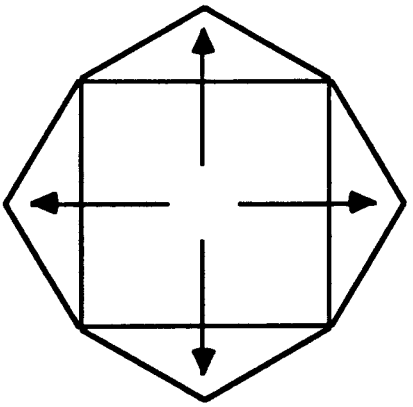


Figure 32 : Récupération des commandes

ii) LA PROPAGATION LIMITEE

La propagation limitée se distingue de la propagation pure sur deux plans :

- le point germe à partir duquel se propage la commande dépend des données, il n'est plus fixe. Cela nécessite la définition d'un protocole de routage pour atteindre la première cellule.
- la diffusion est limitée à une partie du réseau, mais non totalement réduite au tracé lui-même.

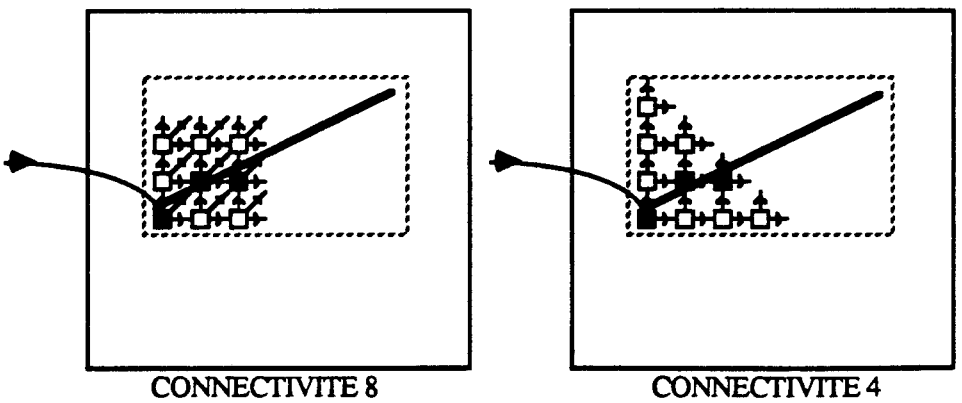


Figure 33 : Propagation limitée

Dans les cellules, les calculs, hormis le test de limitation, ont lieu après la diffusion de la commande.

Deux problèmes se posent lors du développement d'un algorithme à propagation limitée :

- comment atteindre la cellule de départ,
- comment savoir si l'algorithme a été exécuté en totalité.

◊ Le premier problème est résolu par un mécanisme de routage plus ou moins complexe. Ce protocole est utilisé par tous les algorithmes désirant communiquer avec une cellule particulière du réseau.

Exemple de routage trivial sur un réseau classique auquel on accède par un bord :

Pour joindre la cellule (i,j) depuis l'ordinateur hôte il suffit d'envoyer, dans la ligne j, le message M à transmettre précédé du numéro de colonne i de la cellule à atteindre. Chaque cellule teste alors le premier terme reçu et détermine ainsi si la commande lui est destinée ou non. Si le message ne la concerne pas elle le transmet à sa voisine :

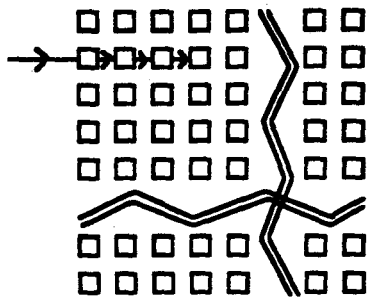


Figure 34 : Routage simple

C'est le routage le plus élémentaire qui soit.

Un tel mécanisme ne résoud pas les conflits d'accès. Mais remarquons que si une cellule désire transmettre l'information à une de ses voisines et que celle-ci est active, il n'est pas impensable que la cellule attende purement et simplement que sa voisine termine son exécution car les traitements effectués par les cellules sont de courte durée. Il faut prendre garde à l'interblocage.

Nous ne développerons pas plus les protocoles de routage, car ceux-ci sont fortement liés à l'architecture et aux mécanismes de communication et de synchronisation mis en place. Ils sont d'ailleurs généralement câblés [MAZ 88].

◊ Le second problème est celui du test d'arrêt.

En effet, les cellules ayant un comportement autonome à l'intérieur du réseau, ce dernier n'est pas contrôlable de l'extérieur et il n'est pas facile de déterminer s'il reste ou non des cellules en activité.

Contrairement au problème du routage, plutôt architectural, et commun au divers algorithmes, celui du test d'arrêt, bien qu'intéressant tous ces algorithmes, adopte des solutions différant sensiblement en fonction de l'objet du programme.

Deux solutions générales peuvent néanmoins être considérées :

- le Time-out ou Temps maximal
- le Test d'activité

i) LE TIME-OUT

Quel que soit le travail à exécuter il est possible de borner le temps d'exécution. Après ce laps de temps, la commande est considérée comme étant entièrement exécutée.

Bien évidemment cette solution n'est pas optimale car le temps maximum estimé est souvent très supérieur au temps d'exécution réel.

Cette solution est commune à tous les algorithmes mais il faut quand même évaluer la limite à chaque exécution puisqu'elle varie en fonction des données.

Enfin, cette approche suppose qu'il n'y ait pas d'interférence entre les zones activées par les commandes, cela rendrait l'estimation du temps caduque.

ii) LE TEST D'ACTIVITE

Comme son nom l'indique il consiste à tester s'il reste des cellules actives dans le réseau. Il est donc nécessaire d'effectuer des tests tant qu'une cellule active est détectée, ce qui augmente, plus ou moins fortement suivant la fréquence des tests, le temps global d'exécution.

Pour optimiser quelque peu cette solution, il est possible d'évaluer un temps minimal d'exécution avant lequel aucun test ne sera effectué.

Cette deuxième solution fonctionne bien quand il n'y a qu'une commande en cours. Sinon, un estampillage est nécessaire, comme souvent d'ailleurs, mais ici l'analyse du balayage test est délicate à mettre en place.

C'est une solution qui, pour être efficace doit être câblée. Nous ne la détaillerons pas plus ici.

iii) LE SUIVI DE CONTOUR

Le suivi de contour a pour principale caractéristique de ne s'écarter que très peu du contour, car chaque cellule choisit la voisine à qui elle transmet la commande.

Un protocole est nécessaire pour atteindre la cellule initiale.

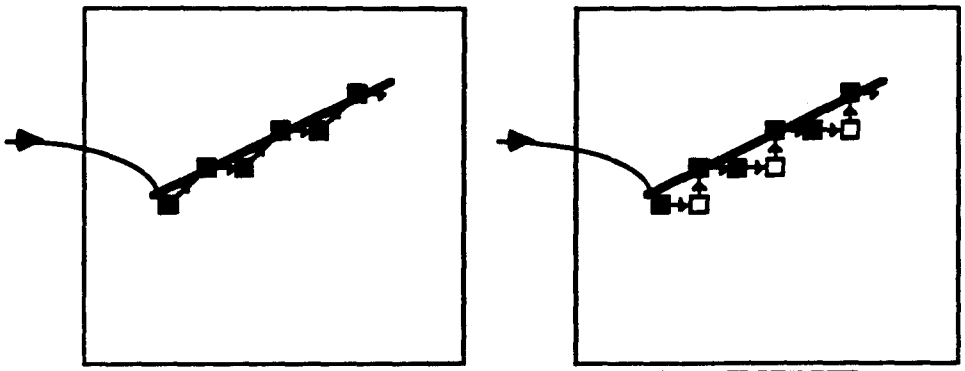


Figure 35 : Suivi de contour

Cette solution a recours par deux fois à un protocole de routage :
- pour atteindre la cellule initiale,
- pour contrôler la totale exécution de la commande.

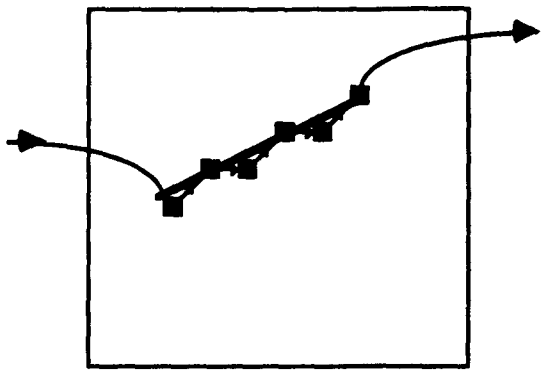


Figure 36 : Suivi de contour

1.4.2. LE PARALLELISME

Nous allons tout d'abord définir un certain nombre de termes permettant de qualifier les types de parallélisme :

- Le pipeline algorithmique : Le calcul du point commence avant la fin du calcul du point précédent.
- Le parallélisme algorithmique : Plusieurs points sont calculés simultanément.
- Le pipeline de commandes : Une commande est émise avant que la commande précédente ne soit terminée.
- Le parallélisme de commandes : Plusieurs commandes sont injectées simultanément dans le réseau.

Les deux derniers types concourent à définir ce que l'on appelle communément le parallélisme objet.

1.4.2.1. DIFFUSION TOTALE

C'est la solution S.I.M.D.

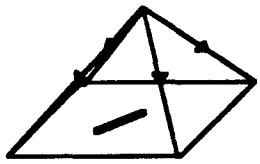


Figure 37 : Diffusion à tout le réseau

Le seul type de parallélisme présent dans ce mode de fonctionnement est le parallélisme algorithmique qui est optimum puisqu'il atteint N^2. En effet, tous les points sont calculés en parallèle. Le rendement n'en est pas pour autant important étant donné le faible nombre de cellules réellement utiles. Il est de l'ordre de n/N^2 (n : Nombre de points à tracer; N : Taille du réseau).

Le parallélisme objet est nul, une seule commande est exécutée à la fois, car chacune d'entre-elles requiert l'activation totale du réseau.

1.4.2.2. MULTIPIPELINES

Les architectures multipipelines en présence sont les suivantes :

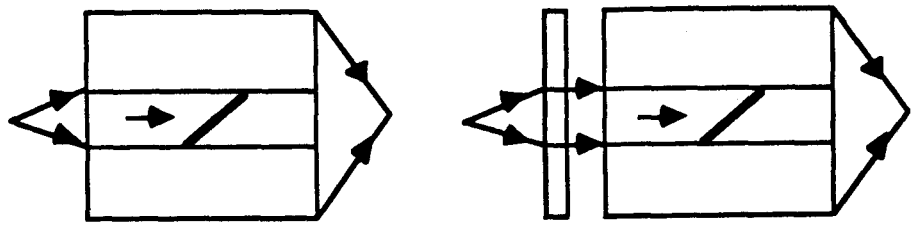


Figure 38 : Architectures multipipelines

Tous les types de parallélisme sont présents à un degré plus ou moins élevé dans cette architecture :

- Le pipeline algorithmique est important en regard du programme exécuté par la cellule. En effet, la transmission de la commande à la cellule voisine se fait après la simple incrémentation d'un de ses paramètres.

- Le parallélisme algorithmique est quant à lui assez faible. Simplement les points situés sur une même colonne sont tracés simultanément.

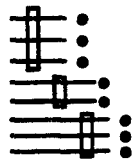


Figure 39 : Parallélisme algorithmique

- Le pipeline de commandes est plus significatif :

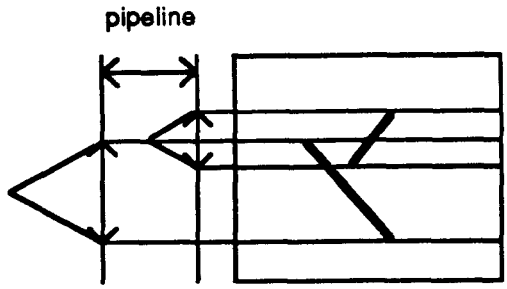


Figure 40 : Pipeline de commandes

Il n'est pas nécessaire d'attendre que la première commande soit exécutée pour commander le tracé du second.

- Le parallélisme de commandes n'est pas non plus à négliger :

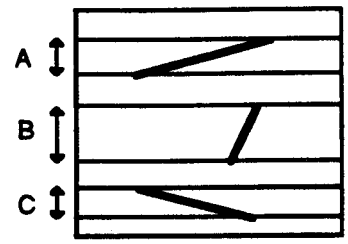


Figure 41 : Parallélisme de commandes

A, B et C peuvent être tracés simultanément.

Pour une utilisation optimale de cette solution il faut mettre en place une stratégie de répartition des objets. Cet algorithme d'ordonnancement, exécuté sur la machine hôte, ne fait pas partie des thèmes abordés dans cette étude.

1.4.2.3. ACCES UNIQUE

Trois approches sont possibles :

- Propagation pure : Les cellules n'effectuent pas de calcul avant de propager la commande.
- Propagation limitée : La commande est transmise à des cellules n'appartenant pas au tracé, mais relativement proches de celui-ci. Le test de limitation, seul calcul effectué avant la transmission, est simple autorisant un pipeline algorithmique important.
- Propagation à suivi de contour : la direction de propagation est fonction du traitement effectué par la cellule. De plus, la commande n'est transmise qu'aux cellules utiles, celles faisant partie du tracé.

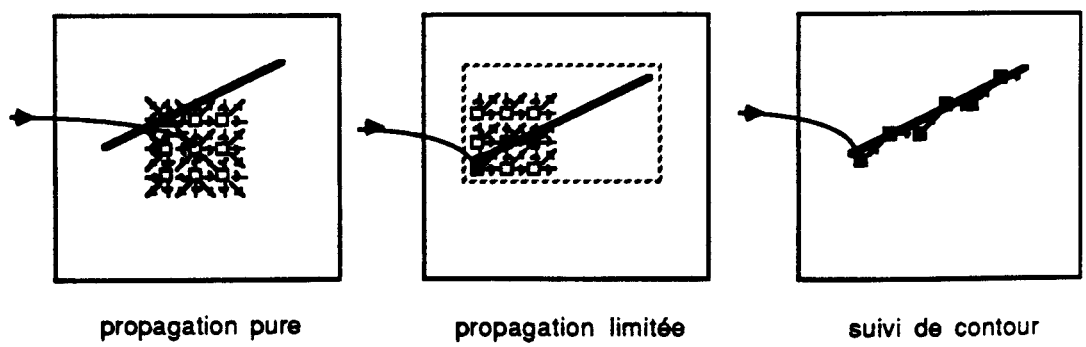


Figure 42 : Différents types de propagations

1.4.2.3.1. LA PROPAGATION PURE

C'est une solution architecturale essentiellement pipeline :
- au niveau algorithmique, puisque la commande est transmise pratiquement dès réception,
- au niveau commandes. A un moment donné, N/2 commandes sont en cours d'exécution.

1.4.2.3.2. LA PROPAGATION LIMITEE

La propagation limitée offre un taux de parallélisme objet élevé dû à la faible importance de la zone activée, mais l'optimisation de cette solution requiert un équilibrage de charge non trivial à réaliser.

Le parallélisme algorithmique est symbolique, puisque seuls un maximum de trois à cinq points par germe (en fonction de la connectivité choisie), sont calculés en parallèle.

1.4.2.3.3. LE SUIVI DE CONTOUR

L'exécution de l'algorithme est quasiment séquentielle, seul l'affichage qui a lieu après transmission introduit un peu de pipeline algorithmique.

Son principal attrait est le nombre limité de cellules activées pour un tracé, autorisant ainsi un parallélisme objet particulièrement important. En revanche cela donne naissance à des conflits d'accès entre commandes différentes nécessitant une gestion fine des communications.

1.4.3. NOTES SUR LA MACHINE HOTE

Les algorithmes qui sont présentés tout au long de cette étude, font appel à un ordinateur hôte.

Cela suppose donc l'existence d'une machine assez performante pour ne pas nuire à l'efficacité du réseau.

Néanmoins, dans le cas où l'ordinateur hôte est insuffisant, il est possible de reporter le travail qui lui était confié, dans tout ou partie des cellules du réseau. Ce travail est constitué principalement de l'évaluation des paramètres des commandes.

- 1ère solution :

  Reporter le préprogramme effectué auparavant par l'ordinateur hôte dans chacune des cellules ce qui aboutit à un fonctionnement SIMD.

- 2ème solution :

  Le report du travail dans la cellule de départ.

  avantage : Les tâches sont réparties au travers du réseau, puisque la cellule initiale est généralement définie en fonction des données.

  inconvénient : D'une part le nombre de commandes susceptibles d'être interprété par une cellule est doublé (cellule initiale ou non), d'autre part leur travail est plus conséquent.

Remarque :

Cette deuxième solution conduit à des architectures spécialisées, dans le cas d'un fonctionnement multipipeline :

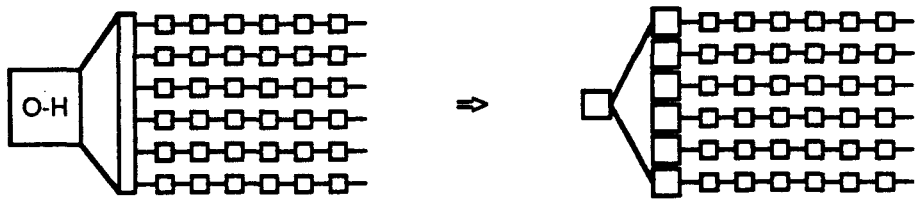


Figure 43 : Architecture multipipeline particulière

Avec ce type de fonctionnement, seule la première cellule de la ligne recevant la commande exécute le préprogramme puis transmet une commande avec des paramètres préévalués.

Le choix du meilleur compromis est fonction des algorithmes à implémenter et des possibilités techniques du moment.

1.5. LES NOTATIONS UTILISEES

Bien que dans cette étude les protocoles de routage ne soient pas étudiés, il est indispensable de définir un formalisme concernant les instructions permettant de communiquer avec les cellules :

- Pour les communications depuis l'ordinateur hôte, qu'elles soient directes ou avec protocole, nous utiliserons l'instruction :

  Envoi à cellule (X ,Y) de (Commande)

  où X et Y sont les coordonnées de la cellule destinataire

- Pour les communications directes, d'une cellule à sa voisine, l'instruction est :

Transmission (I , J) de (Commande)

où I et J prennent leur valeur dans (-1 , 0 , 1)

Le couple (I , J) indique la direction de transmission :

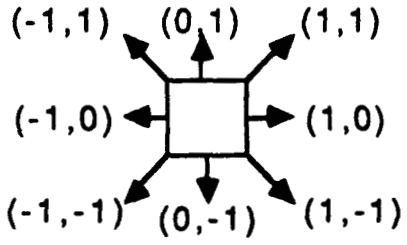


Figure 44 : Codage des directions d'émission

- L'instruction de réception, quant à elle, est simplement :

Réception de (Commande)

Remarques :

- Etant donné leur diversité, il n'y a pas d'instruction permettant des envois multiples. Elles seront détaillées à chaque fois que nous y ferons appel. Dans la pratique il est préférable de s'en passer, à cause des problèmes de synchronisation engendrés.

- Les cellules sont repérées par référence au pixel auquel elles sont associées, l'écran étant lui-même généralement repéré à partir du point inférieur gauche de l'image :

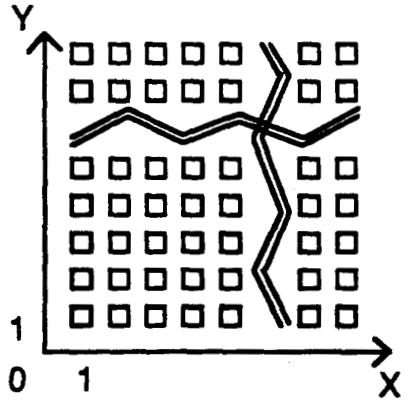


Figure 45 : Numérotation des cellules

BIBLIOGRAPHIE :

[ABJ 85], [ARO 86], [ARO 87], [BAT 80], [BHR 88], [CML 87], [COD 68], [FOU 88], [FRI 83], [FUC 81], [FUC 85], [FUC 88], [FWA 87], [GAL 87], [GHL 88], [GOU 80], [HEM 87], [HFU 83], [KID 83], [KUN 82], [KUN 87], [LMA 86], [MAZ 88], [MBM 87], [NEU 66], [PAB 84], [PIE 88], [PHO 75], [PRD 84], [POT 83], [RGZ 85], [ROS 83], [SEI 85], [SHS 87], [WOL 86].

2. LE TRACE DE SEGMENTS

TABLE DES MATIERES

2. LE TRACE DE SEGMENTS

2.1. GENERALITES

[HEG 85] [NWS 81] [PER 88]

L'utilisation d'une matrice de pixels pour visualiser une scène pose un certain nombre de problèmes, parmi lesquels celui du tracé de segments.

L'énoncé en est le suivant : connaissant les extrémités du segment à tracer, déterminer quel est la suite de pixels s'approchant le plus du segment théorique ou plus formellement : trouver le segment *discret* approchant le segment *continu*.

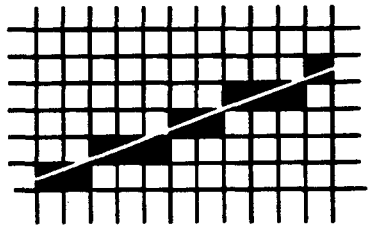


Figure 1 : Représentation dans l'espace discret

La recherche d'un algorithme de tracé de segment performant, est primordiale lors de la réalisation d'un logiciel graphique de synthèse d'images, étant donné sa fréquence d'utilisation..

En effet, la visualisation du segment sur le plan discret que constitue l'image est utile, non seulement pour la conception de dessins filaires, mais également pour l'affichage d'objets constitués de facettes polygonales, ainsi que pour d'autres problèmes pour lesquels il n'y a pas d'affichage, mais où le calcul des points est néanmoins indispensable, par exemple lorsqu'il s'agit de déterminer le point d'intersection de deux droites.

Hormis les problèmes de rendu réaliste, le tracé de segments est un des plus gourmand en temps de calcul, non par sa complexité mais plutôt par le nombre d'utilisations. Cet usage intensif est un des points qui justifie l'approche massivement parallèle pour la synthèse d'images.

Un tracé correct satisfait à quelques obligations :

- le segment obtenu doit apparaître rectiligne,

Figure 2 : Tracé correct, Tracé incorrect

- il doit s'arrêter au point désiré (ce qui n'est pas toujours le cas) :

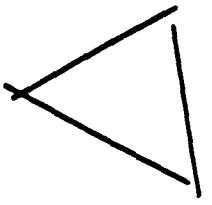


Figure 3 : Point d'arrêt

- la densité du tracé doit rester constante :

  La densité est le nombre de pixels exprimé en fonction de la longueur. Il ne faut pas qu'apparaissent des grosseurs :

Figure 4 : Densité constante

- enfin, cette densité doit être indépendante de la longueur et de la pente du segment, de façon à ne pas avoir de tracés plus épais que d'autres :

Figure 5 : Densité indépendante de la pente

Ces critères ne sont pas toujours respectés dans leur intégralité.

Même l'algorithme de BRESENHAM, universellement reconnu, ne satisfait pas le dernier point. En effet, un segment horizontal est de densité égale à 1, alors que celle d'un segment oblique à 45° est de :

$$d = \frac{\text{Nb de pixels}}{\text{Longueur}} = \frac{n}{(n\sqrt{2})} = \frac{\sqrt{2}}{2}$$

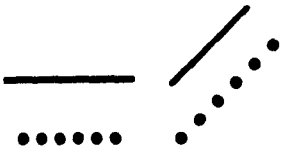


Figure 6 : Tracés de BRESENHAM

En fait, pour obtenir des tracés de qualité irréprochable, l'apport des techniques d'antialiassage est indispensable. Notre propos n'est pas étendu à ce problème.

La plupart des algorithmes de tracé de segments qui ont été développés sont de type incrémental. Autrement dit, chaque point se déduit du point précédent.

D'autre part, ces algorithmes ont comme caractéristiques de pouvoir se suffir de calculs sur des nombres entiers et de ne pas avoir recours aux multiplications, ce qui permet d'envisager de les câbler.

Cette dernière particularité est très importante car un algorithme câblé donne des performances sans commune mesure avec sa réalisation logicielle.

2.2. LES ALGORITHMES SEQUENTIELS

2.2.1. L'ALGORITHME DE BRESENHAM

[BRE 67] [DUR 83] [PEL 85]

C'est l'algorithme le plus fréquemment utilisé.

De nombreuses variantes ont été développées sur le principe de l'algorithme proposé par BRESENHAM, mettant à profit telle ou telle particularité du tracé, comme la symétrie par exemple.

BRESENHAM propose un découpage du plan en octants, qui donne naissance à huit mouvements différents.

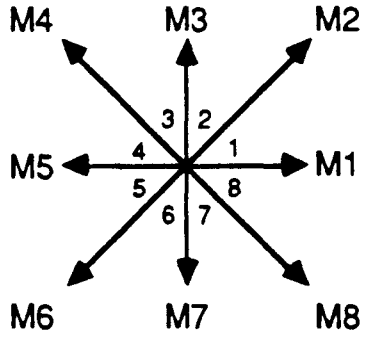


Figure 7 : Numérotation des octants et des mouvements

Ce découpage est effectué par rapport à l'une des extrémités du segment. Il en résulte que tout segment est inscrit dans un seul octant et ne requiert donc pour son tracé que deux mouvements.

Par exemple, un segment du premier octant ne nécessite que les mouvements M1 et M2.

Figure 8 : Segment du premier octant

Ce découpage avait été utilisé auparavant par FREEMAN [FRE 61] pour en faire une structure de données pour le stockage des segments. Tout segment est conservé sous la forme d'une succession de mouvements unitaires, cela a engendré toute une algorithmique spécifique à cette structure.

Le tracé est obtenu en propageant de point en point l'erreur verticale par rapport au tracé idéal.

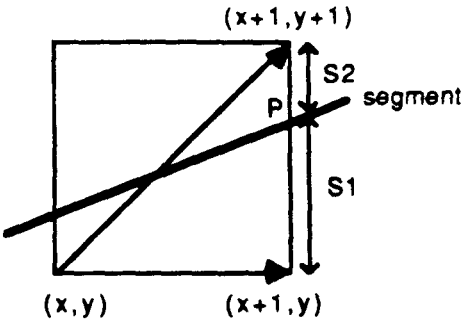


Figure 9 : Evaluation de l'erreur pour BRESENHAM

Suivant le rapport des longueurs des segments S1 et S2, on choisira soit le mouvement axial, soit le mouvement diagonal.

L'algorithme proposé se décompose en deux phases successives :

- déterminer l'octant et les mouvements associés,
- calculer la suite de points.

{ cf ANNEXE A : SEGMENT SEQUENTIEL 1 }

2.2.2. L'ALGORITHME PAR ANALYSE DIFFERENTIELLE OU D.D.A.

[FIE 85] [NWS 81] [PEL 85] [REV 88]

Connaissant un point du segment, son successeur est calculé par simple application d'un pas vertical et d'un pas horizontal. Les incréments horizontaux et verticaux sont calculés par rapport à la plus grande des différences en X ou en Y.

Le choix est tel qu'un des deux incréments est unitaire, l'autre est un réel de valeur absolue inférieure à 1.

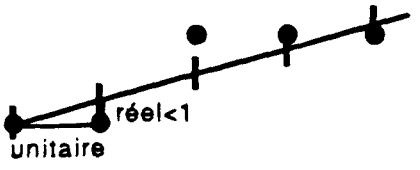


Figure 10 : Tracé DDA

Les coordonnées des points sont exprimées dans l'espace continu, elles sont réelles, l'affichage se fait donc après approximation.

{ cf ANNEXE A : SEGMENT SEQUENTIEL 2 }

Comme pour l'algorithme de BRESENHAM, de nombreuses variantes ont été développées à partir de ce principe, tenant compte notamment de la régularité du tracé.

On peut consulter à ce sujet [DUR 83], [LOC 80]. LOCEFF choisit d'utiliser, en association avec l'incrément unitaire, un incrément réel supérieur en valeur absolue à 1, ce qui nous le verrons, est intéressant pour un réseau cellulaire.

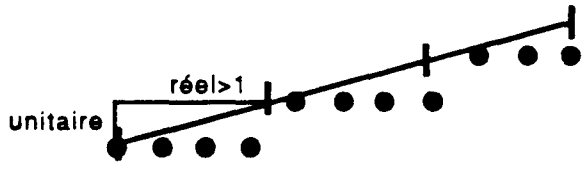


Figure 11 : Tracé de LOCEFF

2.2.3. L'ALGORITHME DICHOTOMIQUE

[PEL 85]
Ce dernier algorithme est fondamentalement différent des précédents. Il est peu utilisé dans les approches séquentielles mais, il peut s'avérer efficace sur une machine parallèle appropriée.

Le principe de l'algorithme dichotomique est simple :

Soient $A(x_a,y_a)$ et $B(x_b,y_b)$ les deux extrémités d'un segment à tracer, le point milieu M de coordonnées $((x_a+x_b)/2,(y_a+y_b)/2)$ appartient au segment S, ce qui donne, après arrondi, un troisième point.

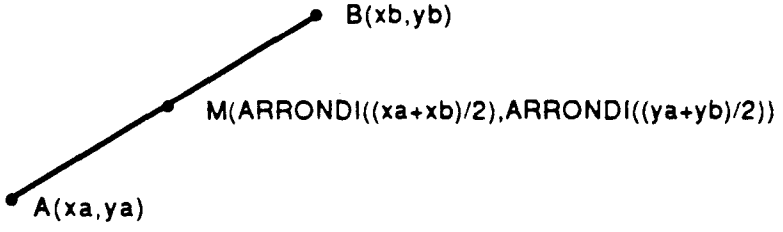


Figure 12 : M milieu de AB

En appliquant le raisonnement aux deux sous-segments AM et MB nous obtenons deux points supplémentaires et quatre sous-segments. Les coordonnées exactes des sous-segments doivent être transmises, et non les points obtenus après arrondi, sous peine d'obtenir un tracé erroné, dû à la propagation de l'erreur commise lors de l'approximation.

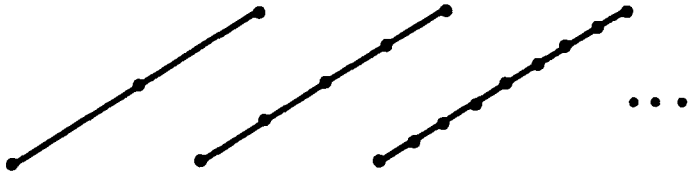


Figure 13 : Tracé dichotomique

En réitérant le processus, le segment S est entièrement affiché par une méthode récursive qui travaille sur des fractionnaires.

{ cf ANNEXE A : SEGMENT SEQUENTIEL 3 }

2.2.4. L'ALGORITHME LOGIQUE

[STA 74] [STA 75]

Pour utiliser cet algorithme, il faut disposer de deux mémoires de la taille de la mémoire d'image.

Reprenons le découpage préconisé par BRESENHAM :

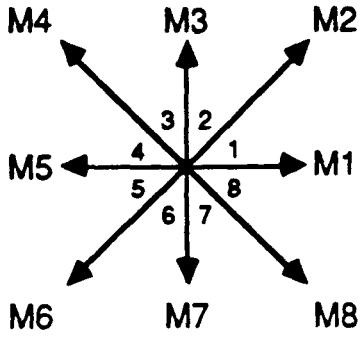


Figure 14 : Numérotation des directions

Le principe de l'algorithme est alors le suivant :

1) Tracer les droites booléennes en chacune des extrémités suivant le couple (direction, direction opposée) (M1,M5)

2) Remplir l'espace booléen ainsi défini

3) Conserver le résultat dans la première mémoire booléenne

4) Répéter les actions 1) et 2) pour les directions (M2,M6) et conserver le résultat dans la deuxième mémoire

5) Effectuer un ET logique entre ces deux mémoires et conserver le résultat dans la première

6) Répéter les actions 4) et 5) pour les couples de directions (M3,M7) et (M4,M8)

2. LE TRACE DE SEGMENTS

TABLE DES MATIERES

2. LE TRACE DE SEGMENTS

2.1. GENERALITES

[HEG 85] [NWS 81] [PER 88]

L'utilisation d'une matrice de pixels pour visualiser une scène pose un certain nombre de problèmes, parmi lesquels celui du tracé de segments.

L'énoncé en est le suivant : connaissant les extrémités du segment à tracer, déterminer quel est la suite de pixels s'approchant le plus du segment théorique ou plus formellement : trouver le segment *discret* approchant le segment *continu*.

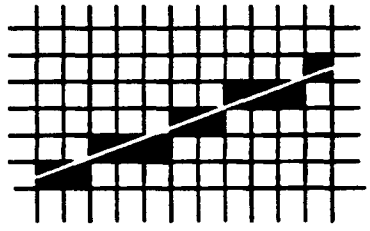


Figure 1 : Représentation dans l'espace discret

La recherche d'un algorithme de tracé de segment performant, est primordiale lors de la réalisation d'un logiciel graphique de synthèse d'images, étant donné sa fréquence d'utilisation..

En effet, la visualisation du segment sur le plan discret que constitue l'image est utile, non seulement pour la conception de dessins filaires, mais également pour l'affichage d'objets constitués de facettes polygonales, ainsi que pour d'autres problèmes pour lesquels il n'y a pas d'affichage, mais où le calcul des points est néanmoins indispensable, par exemple lorsqu'il s'agit de déterminer le point d'intersection de deux droites.

Hormis les problèmes de rendu réaliste, le tracé de segments est un des plus gourmand en temps de calcul, non par sa complexité mais plutôt par le nombre d'utilisations. Cet usage intensif est un des points qui justifie l'approche massivement parallèle pour la synthèse d'images.

Un tracé correct satisfait à quelques obligations :

- le segment obtenu doit apparaître rectiligne,

Figure 2 : Tracé correct, Tracé incorrect

- il doit s'arrêter au point désiré (ce qui n'est pas toujours le cas) :

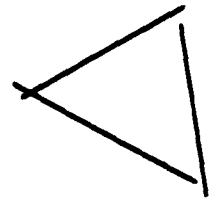


Figure 3 : Point d'arrêt

- la densité du tracé doit rester constante :

  La densité est le nombre de pixels exprimé en fonction de la longueur. Il ne faut pas qu'apparaissent des grosseurs :

Figure 4 : Densité constante

- enfin, cette densité doit être indépendante de la longueur et de la pente du segment, de façon à ne pas avoir de tracés plus épais que d'autres :

Figure 5 : Densité indépendante de la pente

Ces critères ne sont pas toujours respectés dans leur intégralité.

Même l'algorithme de BRESENHAM, universellement reconnu, ne satisfait pas le dernier point. En effet, un segment horizontal est de densité égale à 1, alors que celle d'un segment oblique à 45° est de :

$$d = \frac{\text{Nb de pixels}}{\text{Longueur}} = \frac{n}{(n\sqrt{2})} = \frac{\sqrt{2}}{2}$$

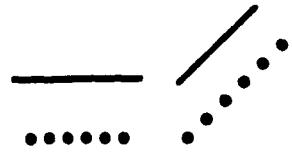


Figure 6 : Tracés de BRESENHAM

En fait, pour obtenir des tracés de qualité irréprochable, l'apport des techniques d'antialiassage est indispensable. Notre propos n'est pas étendu à ce problème.

La plupart des algorithmes de tracé de segments qui ont été développés sont de type incrémental. Autrement dit, chaque point se déduit du point précédent.

D'autre part, ces algorithmes ont comme caractéristiques de pouvoir se suffir de calculs sur des nombres entiers et de ne pas avoir recours aux multiplications, ce qui permet d'envisager de les câbler.

Cette dernière particularité est très importante car un algorithme câblé donne des performances sans commune mesure avec sa réalisation logicielle.

2.2. LES ALGORITHMES SEQUENTIELS

2.2.1. L'ALGORITHME DE BRESENHAM

[BRE 67] [DUR 83] [PEL 85]

C'est l'algorithme le plus fréquemment utilisé.

De nombreuses variantes ont été développées sur le principe de l'algorithme proposé par BRESENHAM, mettant à profit telle ou telle particularité du tracé, comme la symétrie par exemple.

BRESENHAM propose un découpage du plan en octants, qui donne naissance à huit mouvements différents.

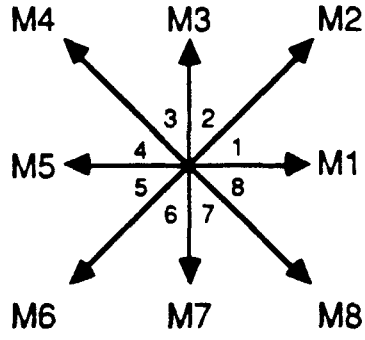


Figure 7 : Numérotation des octants et des mouvements

Ce découpage est effectué par rapport à l'une des extrémités du segment. Il en résulte que tout segment est inscrit dans un seul octant et ne requiert donc pour son tracé que deux mouvements.

Par exemple, un segment du premier octant ne nécessite que les mouvements M1 et M2.

Figure 8 : Segment du premier octant

Ce découpage avait été utilisé auparavant par FREEMAN [FRE 61] pour en faire une structure de données pour le stockage des segments. Tout segment est conservé sous la forme d'une succession de mouvements unitaires, cela a engendré toute une algorithmique spécifique à cette structure.

Le tracé est obtenu en propageant de point en point l'erreur verticale par rapport au tracé idéal.

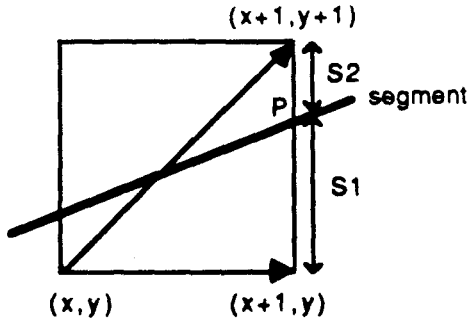


Figure 9 : Evaluation de l'erreur pour BRESENHAM

Suivant le rapport des longueurs des segments S1 et S2, on choisira soit le mouvement axial, soit le mouvement diagonal.

L'algorithme proposé se décompose en deux phases successives :

- déterminer l'octant et les mouvements associés,
- calculer la suite de points.

{ cf ANNEXE A : SEGMENT SEQUENTIEL 1 }

2.2.2. L'ALGORITHME PAR ANALYSE DIFFERENTIELLE OU D.D.A.

[FIE 85] [NWS 81] [PEL 85] [REV 88]

Connaissant un point du segment, son successeur est calculé par simple application d'un pas vertical et d'un pas horizontal. Les incréments horizontaux et verticaux sont calculés par rapport à la plus grande des différences en X ou en Y.

Le choix est tel qu'un des deux incréments est unitaire, l'autre est un réel de valeur absolue inférieure à 1.

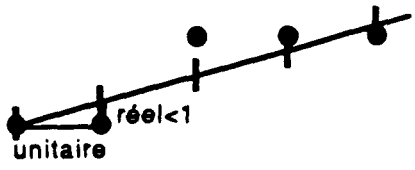


Figure 10 : Tracé DDA

Les coordonnées des points sont exprimées dans l'espace continu, elles sont réelles, l'affichage se fait donc après approximation.

{ cf ANNEXE A : SEGMENT SEQUENTIEL 2 }

Comme pour l'algorithme de BRESENHAM, de nombreuses variantes ont été développées à partir de ce principe, tenant compte notamment de la régularité du tracé.

On peut consulter à ce sujet [DUR 83], [LOC 80]. LOCEFF choisit d'utiliser, en association avec l'incrément unitaire, un incrément réel supérieur en valeur absolue à 1, ce qui nous le verrons, est intéressant pour un réseau cellulaire.

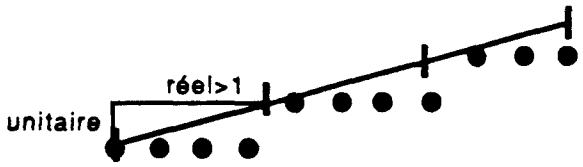


Figure 11 : Tracé de LOCEFF

2.2.3. L'ALGORITHME DICHOTOMIQUE

[PEL 85]
Ce dernier algorithme est fondamentalement différent des précédents. Il est peu utilisé dans les approches séquentielles mais, il peut s'avérer efficace sur une machine parallèle appropriée.

Le principe de l'algorithme dichotomique est simple :

Soient $A(x_a,y_a)$ et $B(x_b,y_b)$ les deux extrémités d'un segment à tracer, le point milieu M de coordonnées $((x_a+x_b)/2,(y_a+y_b)/2)$ appartient au segment S, ce qui donne, après arrondi, un troisième point.

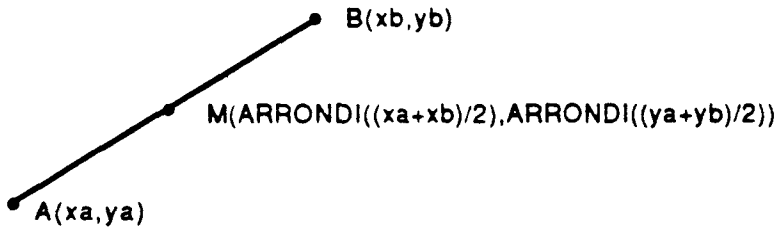


Figure 12 : M milieu de AB

En appliquant le raisonnement aux deux sous-segments AM et MB nous obtenons deux points supplémentaires et quatre sous-segments. Les coordonnées exactes des sous-segments doivent être transmises, et non les points obtenus après arrondi, sous peine d'obtenir un tracé erroné, dû à la propagation de l'erreur commise lors de l'approximation.

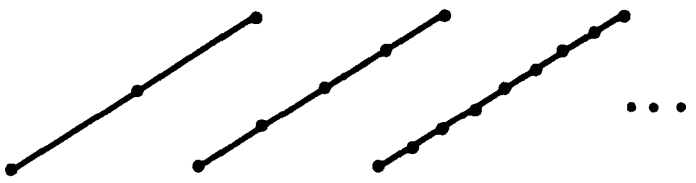


Figure 13 : Tracé dichotomique

En réitérant le processus, le segment S est entièrement affiché par une méthode récursive qui travaille sur des fractionnaires.

{ cf ANNEXE A : SEGMENT SEQUENTIEL 3 }

2.2.4. L'ALGORITHME LOGIQUE

[STA 74] [STA 75]

Pour utiliser cet algorithme, il faut disposer de deux mémoires de la taille de la mémoire d'image.

Reprenons le découpage préconisé par BRESENHAM :

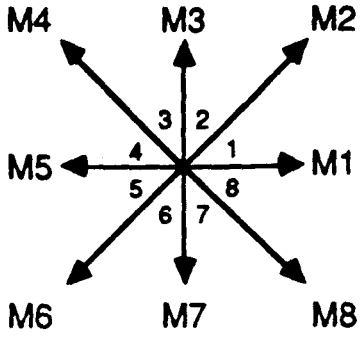


Figure 14 : Numérotation des directions

Le principe de l'algorithme est alors le suivant :

1) Tracer les droites booléennes en chacune des extrémités suivant le couple (direction, direction opposée) (M1,M5)

2) Remplir l'espace booléen ainsi défini

3) Conserver le résultat dans la première mémoire booléenne

4) Répéter les actions 1) et 2) pour les directions (M2,M6) et conserver le résultat dans la deuxième mémoire

5) Effectuer un ET logique entre ces deux mémoires et conserver le résultat dans la première

6) Répéter les actions 4) et 5) pour les couples de directions (M3,M7) et (M4,M8)

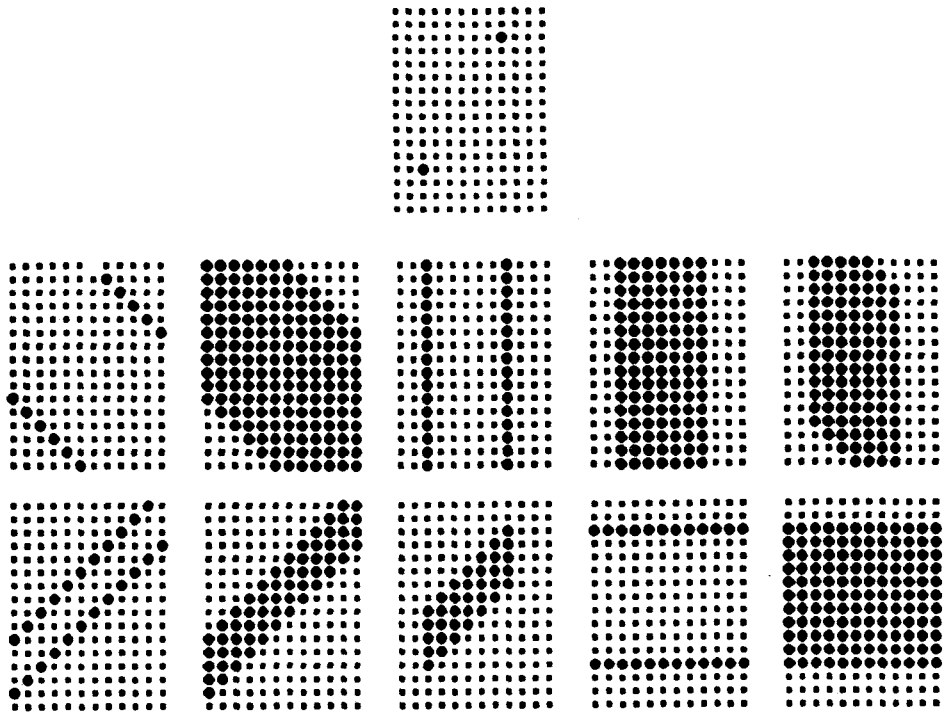


Figure 15 : Elimination dans les huit directions

Il ne reste plus alors qu'à appliquer, suivant chacune des directions, la procédure d'élimination des points qui satisfont la condition.

Cette dernière peut être définie comme suit :
Eliminer les points "noirs" qui ont 4 points "blancs" et 3 ou 4 points "noirs" comme voisins.

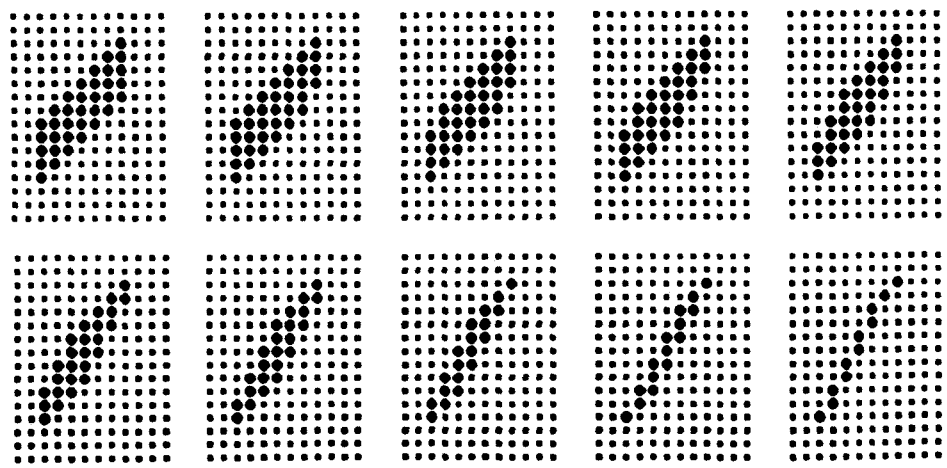


Figure 16 : Phase d'examen du voisinage

Cet algorithme, qui semble intéressant de prime abord, ne trouve pas d'application cellulaire satisfaisante sur le type de réseau que nous étudions. Les raisons en sont premièrement, la procédure d'examen qui nécessite un parcours du voisinage de chaque cellule, coûteux en terme de communications, deuxièmement, le tracé même des droites booléennes et le remplissage qui s'en suit, gourmand en occupation du réseau.

2.3. LES ALGORITHMES PARALLELES.

2.3.1. ALGORITHMES BASES SUR L'EQUATION DE LA DROITE

Dans le chapitre précédent ont été présentées diverses architectures. Chacune d'entre elles engendre un algorithme particulier tenant compte du mode de contrôle et de commande.

2.3.1.1. DIFFUSION TOTALE

Rappel :
Le travail de l'ordinateur hôte consiste simplement à transmettre à chacune des cellules la même commande dont il a évalué les paramètres.

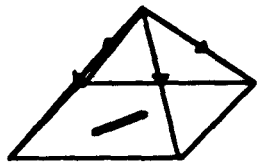


Figure 17 : Distribution de l'équation

Appliqué au tracé de segment, cela conduit à ce que chaque cellule détermine si le pixel qui lui est associé appartient ou non au segment à tracer.

Rappel :
L'équation d'une droite D passant par les points $A(x_a,y_a)$ et $B(x_b,y_b)$ s'exprime sous la forme :

$$D : \{ (x,y) / -\Delta y * x + \Delta x * y + \Delta y * x_a - \Delta x * y_a = 0 \}$$

où $\Delta x = x_a - x_b$ et $\Delta y = y_a - y_b$

L'erreur E est donc :

$$E = -\Delta y * x + \Delta x * y + \Delta y * x_a - \Delta x * y_a$$

Elle est à comparer au seuil d'affichage S qui est quant à lui égal à :

$$(SUP\ (ABS(\Delta x), ABS(\Delta y))) \ DIV\ 2$$

Ce seuil prend en compte l'approximation induite par le travail dans l'espace discret.

Les coefficients de l'équation, ainsi que le seuil d'erreur tolérée sont transmis aux cellules. Chaque cellule vérifie alors si ses propres coordonnées conviennent et dans l'affirmative affiche 'son' pixel.

Il ne reste plus qu'à limiter l'affichage au segment proprement dit :

$$((x_a <= i <= x_b) \text{ ou } (x_b <= i <= x_a))$$

$$\text{et } ((y_a <= j <= y_b) \text{ ou } (y_b <= j <= y_a)))$$

où i= n° de colonne, j = n° de ligne

{ cf ANNEXE A : SEGMENT PARALLELE 1 }

C'est l'algorithme le plus général, il ne tient pas compte de la manière dont l'information est diffusée dans le réseau, ou plutôt il suppose qu'un envoi général vers toutes les cellules du réseau est effectué. Nous reviendrons sur ce point à la fin de ce chapitre (cf 2.4.2).

2.3.1.2. MULTIPIPELINES

Rappel :

Dans les informations se déplacent unilatéralement de la gauche vers la droite.

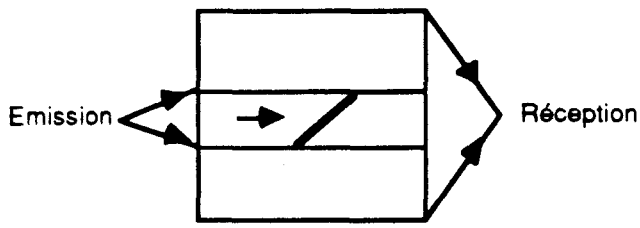


Figure 18 : Multipipeline simple

En fait le test de limitation est désormais divisé en deux :

$$((x_a <= i <= x_b) \text{ ou } (x_b <= i <= x_a))$$

$$\text{et } ((y_a <= j <= y_b) \text{ ou } (y_b <= j <= y_a))$$

où i= n° de colonne, j = n° de ligne
et $((x_a,y_a),(x_b,y_b))$: segment

L'ordinateur hôte ne transmet la commande qu'aux cellules concernées (i.e. celles vérifiant la deuxième partie du test). Chaque cellule n'effectue plus alors que le premier test, d'où un gain de temps appréciable.

Comme pour la solution précédente, le calcul des coefficients de l'équation est commun à toutes les cellules, il est confié à la machine hôte, ce qui limite le travail respectif de chaque cellule.

D'autre part, il est intéressant de remarquer que l'erreur d'une cellule, par rapport à la droite à tracer, se déduit de l'erreur de la cellule précédente :

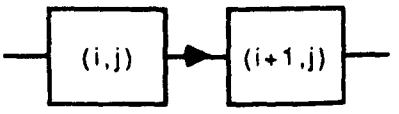


Figure 19 : Transmission de l'erreur

$$E(i,j) = -\Delta_y * i + \Delta_x * j + C$$

$$E(i+1,j) = -\Delta_y * (i+1) + \Delta_x * j + C = E(i,j) - \Delta_y$$

La valeur qui est comparée au seuil varie de Δy à chaque transmission.

{ cf ANNEXE A : SEGMENT PARALLELE 2 }

Sur ce type de réseau, avec ce genre de propagation, il est possible de réduire encore le travail des cellules en utilisant un VECTEUR D'ENTREES.

Rappel :

Au lieu de transmettre une même information à toutes les cellules du réseau, chaque cellule reçoit la commande avec des paramètres spécifiques.

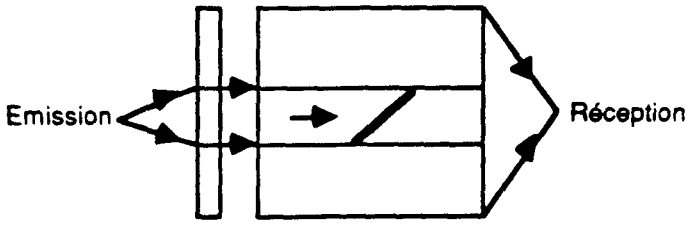


Figure 20 : Multipipeline à vecteur d'entrées

Pour bénéficier pleinement de la remarque faite plus haut quant à la transmission de l'erreur, il faut utiliser un vecteur d'entrées et calculer l'erreur initiale pour chacune des lignes :

$$E(0,j) = -\Delta_y * 0 + \Delta_x * j + C = \Delta_x * j + C$$

Comme pour l'algorithme précédent c'est E-C qui est transmis, donc le vecteur initial est tel que

$$V(j) = (SEGMENT3,\ \Delta_x * j + C,\ \Delta_y,\ S)$$

où $((y_a <= j <= y_b)$ ou $(y_b <= j <= y_a))$

SEGMENT3 : nom de la commande
$\Delta_x * j + C$: erreur initiale
S : seuil d'affichage

Chaque cellule se contente alors d'incrémenter l'erreur avant de la transmettre à sa voisine, puis teste si elle-même fait partie du tracé.

{ cf ANNEXE A : SEGMENT PARALLELE 3 }

Cet algorithme, comme le précédent utilise une arithmétique entière.

Si on s'autorise l'usage des fractionnaires, il est possible de réduire encore le nombre de paramètres à transmettre.

En effet, l'équation de la droite peut se mettre sous la forme :

$$x = A*y + B$$

où $A=(x_a-x_b)/(y_a-y_b)$ et $B= x_a - a*y_a$

L'erreur est :

$$E(i,j) = i - A*j - B$$

Lors d'une transmission l'erreur est incrémentée de 1.

$$E(i+1,j) = E(i,j) + 1$$

D'autre part, l'erreur initiale est :

$$E(0,j) = A*j - B$$

Le seuil S devient :

$$S = SUP(ABS(A),1)/2$$

Donc le vecteur d'entrées V est tel que :

V(j) = (SEGMENT4, -A*j + B, S)

SEGMENT4 : nom de la commande
A*j + B : erreur initiale
S : seuil d'affichage

{ cf ANNEXE A : SEGMENT PARALLELE 4 }

Remarque :

L'utilisation d'un algorithme opérant sur des fractionnaires dépend de la capacité des cellules, même si le programme est plus simple il est souvent préférable de manipuler des entiers.
De plus, un fractionnaire nécessite plus de place pour sa mémorisation, ce qui entraîne une perte de performances lors du transfert entre cellules.
Notons néanmoins que dans le cas qui nous préoccupe il s'agit simplement d'incrémenter ledit fractionnaire.

2.3.1.3. PROPAGATIONS

Rappel :
Pour ce type d'architecture il s'agit de propager l'information à partir d'une cellule du réseau.

Nous avons vu trois approches différentes :

i) LA PROPAGATION PURE

Rappel :
La propagation pure consiste à diffuser, à partir de la cellule centrale, la commande dans tout le réseau.

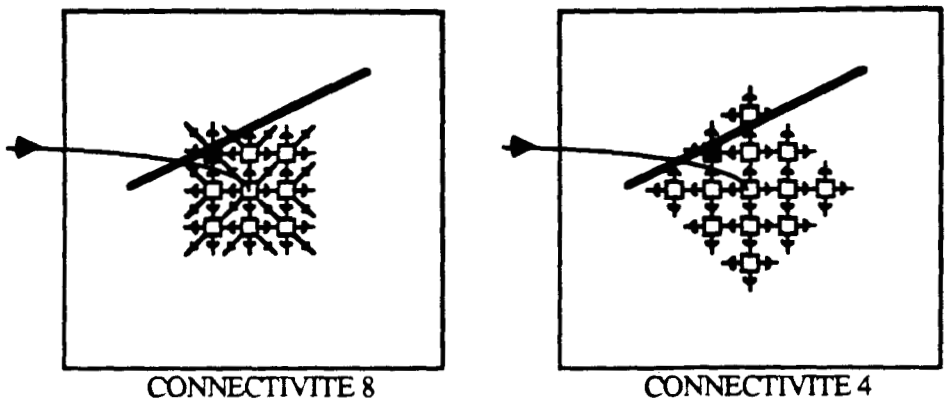


Figure 21 : Propagation pure

Chaque cellule transmet la commande avant d'effectuer le traitement.

Nous développons un algorithme à propagation pure basé sur la transmission de l'équation de la droite support.

Rappelons que l'équation de la droite est de la forme :

$$\{ (i,j) / -\Delta_y * i + \Delta_x * j + C = 0 \}$$

où $\Delta_x = x_b - x_a$, $\Delta_y = y_b - y_a$ et $C = \Delta_y * x_a - \Delta_x * y_a$

Ceci pour le tracé du segment $((x_a,y_a) , (x_b,y_b))$.

Définissons la loi concernant la transmission de l'erreur. Les diverses situations rencontrées sont les suivantes :

$$\begin{aligned}
E(i+1,j) &= E(i,j) - \Delta_y \\
E(i+1,j+1) &= E(i,j) + \Delta_x - \Delta_y \\
E(i,j+1) &= E(i,j) + \Delta_x \\
E(i-1,j+1) &= E(i,j) + \Delta_x + \Delta_y \\
E(i-1,j) &= E(i,j) + \Delta_y \\
E(i-1,j-1) &= E(i,j) - \Delta_x + \Delta_y \\
E(i,j-1) &= E(i,j) - \Delta_x \\
E(i+1,j-1) &= E(i,j) - \Delta_x - \Delta_y
\end{aligned}$$

Le seuil d'affichage quant à lui reste constant et égal à :

$$S = SUP(ABS(\Delta_x), ABS(\Delta_y)) \text{ DIV } 2$$

{ cf ANNEXE A : SEGMENT PARALLELE 5 }

Cet algorithme est appliqué sur le réseau à huit voisins, mais son adaptation à une connectivité quatre est triviale.

ii) LA PROPAGATION LIMITEE

Rappel :

La propagation limitée se fait à partir d'un point germe et est limitée à une partie du réseau

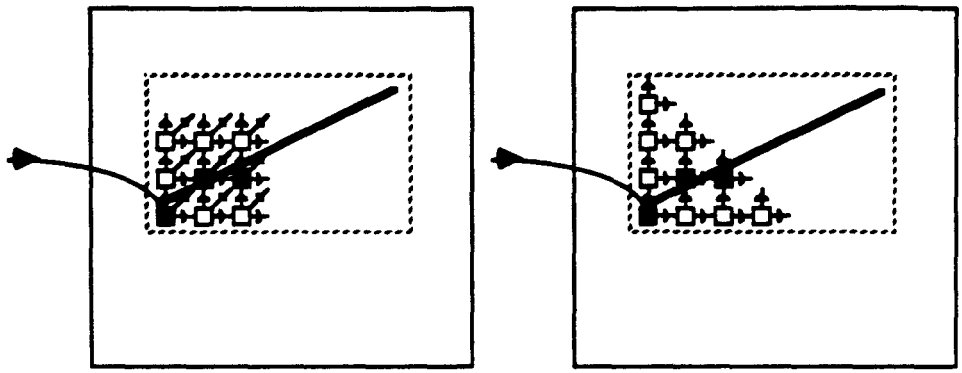


Figure 22 : Propagation limitée

Dans les cellules, les calculs, hormis le test de limitation, ont lieu après la diffusion de la commande.

Dans le cas particulier du segment pour éviter la diffusion complète, il est possible de limiter la propagation aux cellules susceptibles d'appartenir au segment, ceci grâce à un test sur les coordonnées minimales et maximales.

Mais un test plus judicieux sur le seuil permet de limiter la propagation à l'environnement direct du segment.

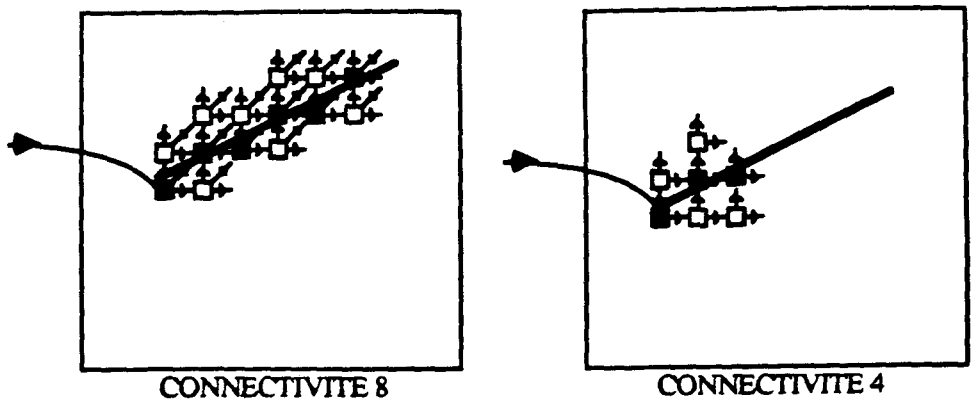


Figure 23 : Propagation limitée par un seuil

Pour que la cellule transmette la commande à ses voisines, l'erreur ne doit pas excéder E_M = ABS(Δx) + ABS(Δy), soit le double du seuil d'affichage.

Ce test est combiné avec celui effectué sur l'abscisse de la cellule.

Par ailleurs, deux points du segment sont connus : ses extrémités.

Ce sera donc à partir de ces deux points que l'algorithme se propagera.

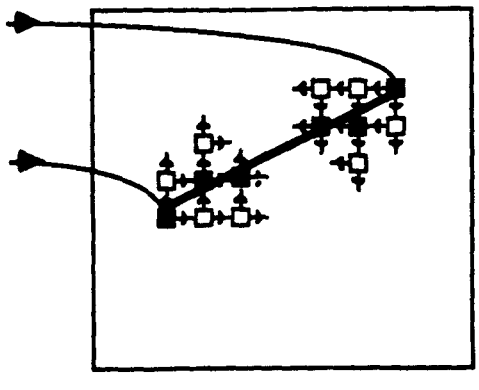


Figure 24 : Propagation à partir de deux germes

La propagation à partir d'autres points caractéristiques, tels que le milieu, augmente le degré de parallélisme, mais entraîne également des calculs supplémentaires dont la complexité, liée aux approximations nécessaires, n'est pas négligeable.

{ cf ANNEXE A : SEGMENT PARALLELE 6 }

iii) LE SUIVI DE CONTOUR

Rappel :

La propagation est limitée au cellule directement concernée par le traitement.

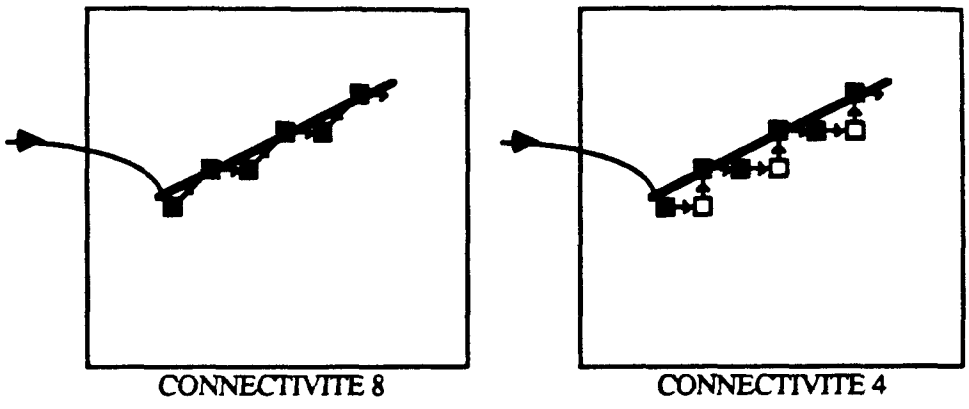


Figure 25 : Suivi de contour

Notons que pour le suivi de contour un réseau de connectivité huit est préférable. Cela évite d'activer des cellules n'appartenant pas au tracé, à seule fin de transmettre la commande, un mini-routage autrement dit.

Ce problème de connectivité est lié à la définition théorique du segment communément admise, celle de BRESENHAM. Cette remarque reste vraie pour le tracé de cercles.

L'étude d'une solution fondée sur l'évaluation de l'équation rejoint les solutions présentées dans les pages qui suivent.

2.3.2. ALGORITHME DE BRESENHAM CELLULAIRE

L'algorithme développé par BRESENHAM présente un caractère éminemment séquentiel puisque chaque point est déterminé en fonction du point précédemment calculé.

Mais il offre l'avantage de ne travailler que sur des entiers sans aucune approximation sans avoir recours aux multiplications, et peut donc être câblé.

La transcription sous forme cellulaire de l'algorithme séquentiel se divise en deux parties distinctes :

- la tâche de l'ordinateur hôte :
  - découpage en octants et désignation des deux directions requises
  - envoi de la commande paramétrée à une des cellules extrémités du segment.

- la tâche de chaque cellule :
  - choisir une des deux directions
  - transmettre la commande.

Il s'agit d'un algorithme à propagation par SUIVI DE CONTOUR.

{ cf ANNEXE A : SEGMENT PARALLELE 7 }

Les paramètres, bien qu'entiers, sont au nombre de huit ce qui nuit aux performances. Pour réduire ce nombre à cinq les couples de directions sont codés, chaque cellule ayant par la suite à interpréter ces codes.

Le nombre de couples de directions différents est ramené à quatre en considérant que le segment est tracé en commençant par l'extrémité d'abscisse la plus petite.

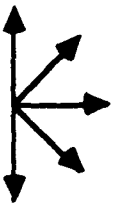


Figure 26 : Segments tracés à partir de l'extrémité la plus proche du bord

{ cf ANNEXE A : SEGMENT PARALLELE 8 }

Remarques communes à ces deux algorithmes :

- Il est possible d'effectuer un traitement plus performant pour les segments parallèles aux axes (ils ne nécessitent qu'un unique mouvement).
- Pour paralléliser l'algorithme, l'ordinateur hôte doit subdiviser le segment en sous-segments. Le problème se pose alors de l'approximation des points extrêmes.
- Il est préférable d'utiliser un réseau carré de cellules à huit voisines. Celui-ci fournira en effet de meilleures performances puisque l'algorithme utilise des déplacements suivant les diagonales.
- Enfin, d'autres versions de ces algorithmes peuvent être développées en utilisant quatre commandes distinctes, suivant la direction du segment. Il y a alors un paramètre en moins à transmettre, les programmes sont plus courts, mais des commandes supplémentaires doivent pouvoir être interprétées par les cellules.

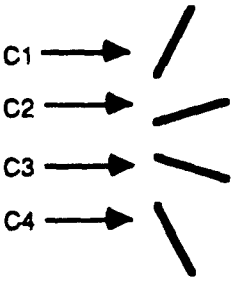


Figure 27 : Commandes différentes suivant l'octant

2.3.3. ALGORITHMES D.D.A. CELLULAIRES

Le principe d'adaptation au réseau de l'algorithme séquentiel est identique à celui employé pour paralléliser l'algorithme de BRESENHAM.

L'ordinateur hôte effectue un précalcul des paramètres puis chaque cellule recevant la commande affiche le pixel qui lui est associé et choisit la cellule vers qui elle va transmettre la commande.

Implémenté tel quel, les paramètres sont au nombre de quatre, des fractionnaires qui plus est. Ce nombre peut être réduit à 2 en remarquant qu'au moins un des incréments est égal, en valeur absolue, à 1. Il ne reste plus alors qu'à transmettre la coordonnée fractionnaire, l'incrément fractionnaire et une variable permettant de déterminer si cela correspond à l'abscisse ou à l'ordonnée. Il est possible de choisir le sens de tracé, tel que l'incrément entier soit toujours égal à +1 (segment non orienté).

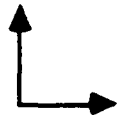


Figure 28 : Orientation des segments pour le DDA

Attention, il ne s'agit plus simplement de tracer le segment à partir de son extrémité d'abscisse la plus petite, il faut aussi tenir compte des valeurs relatives de Δx et Δy.

Si $ABS(\Delta x) > ABS(\Delta y)$ alors il faut propager à partir de l'extrémité d'abscisse la plus faible, sinon il y a propagation à partir du point d'ordonnée la plus petite.

Le comportement est de type suivi de contour.

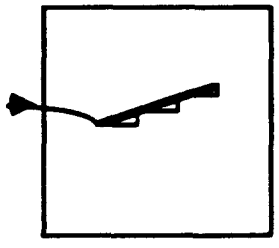


Figure 29 : DDA par suivi de contour

{ cf ANNEXE A : SEGMENT PARALLELE 9 }

Plutôt que d'utiliser un code pour savoir si la valeur qui suit correspond à l'incrément en X ou à celui en Y, il est possible de dédoubler la commande. Il y a ainsi un paramètre en moins à transmettre, mais la cellule doit alors pouvoir interpréter une commande supplémentaire.

Il subsiste néanmoins deux fractionnaires parmi les paramètres ce qui n'est pas toujours acceptable pour les cellules.

A partir de la solution de LOCEFF, nous développons un algorithme quelque peu différent. Rappelons que ce dernier met à profit la régularité du tracé, ce qui équivaut à diviser le segment à tracer en sous-segments horizontaux. Notre idée est d'effectuer sur l'ordinateur hôte la subdivision qui est relativement triviale, et de confier la visualisation des sous-segments au réseau.

Cela se rapproche du multipipeline à vecteur d'entrées. Mais, l'évaluation complète de chacune des commandes est plus longue, en contre partie la tâche des cellules est réduite.

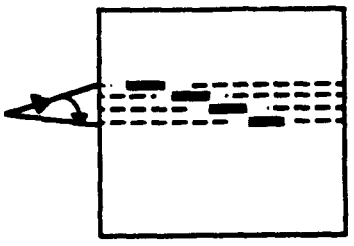


Figure 30 : LOCEFF par multipipeline

{ cf ANNEXE A : SEGMENT PARALLELE 10 }

Remarque :

- Le premier sous-segment du tracé bénéficie d'un traitement particulier, afin de faire coïncider le segment avec la représentation de BRESENHAM.

- Si l'on veut profiter pleinement de l'avantage qu'offre cet algorithme par rapport au DDA classique, il faut pouvoir utiliser deux bords d'émission. C'est la pente qui dicte le choix du bord d'émission. Transit horizontal pour une pente supérieure en valeur absolue à 1, vertical sinon.

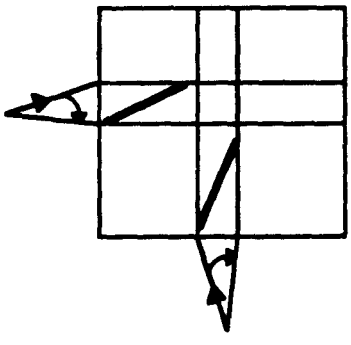


Figure 31 : LOCEFF par double multipipeline

2.3.4. ALGORITHME DICHOTOMIQUE CELLULAIRE

L'énoncé de l'algorithme est des plus simple :

- l'ordinateur hôte envoie à la cellule médiane du segment, les coordonnées de ce dernier,
- chaque cellule recevant cette commande divise le segment en deux, envoie aux cellules milieux des deux demi-segments leurs coordonnées extrêmes, ceci jusqu'à ce que le segment transmis soit réduit à un point.

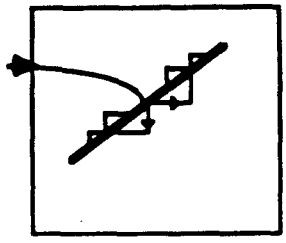


Figure 32 : Solution dichotomique

{ cf ANNEXE A : SEGMENT PARALLELE 11 }

Remarques :

- Le principal problème de cet algorithme est qu'il a recours aux liaisons distantes (i.e. communication entre cellules non directement voisines). Il réclame donc un protocole de routage performant autorisant des cellules distantes à communiquer entre elles.
- Le second problème est qu'il faut effectuer des calculs sur des fractionnaires pour obtenir un tracé correct.
- Il semble qu'un algorithme basé sur ce principe serait plus performant sur une ARCHITECTURE PYRAMIDALE.

2.4. CONCLUSION SUR LES ALGORITHMES CELLULAIRES

I) NOTES SUR LE CRENELAGE

Pour certaines applications une grande qualité de tracé est exigée. Pour ce faire un programme muni d'un mécanisme d'anti-crênelage est nécessaire. Ces algorithmes sont des variantes plus ou moins complexes des algorithmes classiques [FIE 87], [HAN 86].

Remarquons néanmoins que la méthode basée sur l'évaluation de l'équation ne nécessite pas de grande modification pour atténuer le phénomène de crênelage. Il suffit en effet d'effectuer un second test sur l'erreur, permettant ainsi de faire varier l'intensité lumineuse de certains points auparavant rejetés.

II) REMARQUES GENERALES

- Pour chacune des architectures proposées il existe un algorithme parallèle basé sur l'équation du segment.
- Nous vérifierons qu'il est possible de résoudre tout ou au moins la plupart des problèmes de base de la synthèse d'images à l'aide de chacune de ces architectures.
- Si au cours des études qui suivent nous rencontrons une nouvelle architecture, il faudra vérifier qu'il existe un algorithme de tracé de segments adapté.

2.5. TABLEAUX RECAPITULATIFS

Il est particulièrement difficile d'effectuer une comparaison directe tant entre les différents algorithmes, qu'entre les architectures qui leur sont dévolues. En effet, le nombre de paramètres à prendre en compte est très élevé si l'on veut refléter correctement la réalité. De plus, cela suppose un certain nombre d'hypothèses sur la réalisation physique de la machine (hardware), afin de chiffrer les vitesses de transmissions, les temps de calculs, etc...

Néanmoins, nous allons dresser dès à présent quelques tableaux contenant différentes valeurs représentatives des performances attendues.

Remarques :

◊ Nous ne comparons pas toutes les solutions proposées dans ce chapitre, mais seulement celles qui nous semblent dignes d'intérêt :

- L'approche dichotomique n'est pas abordée (elle est plutôt indiquée pour une architecture pyramidale).
- Les solutions à suivi de contour sont étudiées sur des réseaux de connectivité huit (la connectivité 4 utilisant des mini-routages n'est pas reprise).
- La propagation limitée n'est faite qu'à partir d'un point seul germe et elle est limitée par le rectangle englobant. Ceci par souci de cohérence avec les algorithmes étudiés par la suite.

◊ L'hypothèse est faite que les transmissions sont exécutées en parallèle par rapport aux calculs. Cela a pour effet d'introduire des majorants dans les expressions d'évaluation, indiquant si c'est la transmission ou le calcul qui ralentit l'exécution.

◊ En général, on admet que le traitement d'un fractionnaire nécessite deux fois plus de temps que celui d'un entier. Mais, il est possible au prix d'une complexité architecturale supérieure de mettre sur un pied d'égalité les deux traitements. Cette solution n'est rentable que s'il existe une nette prédominence des manipulations de fractionnaires.

Pour notre évaluation nous avons besoin d'une base de référence, nous choisissons la solution la plus classique en nous référant aux entiers.

Rappels : Numérotation des algorithmes

1 : SIMD (équation)
2 : MULTIPIPELINE (équation)
3 : MULTIPIPELINE A VECTEUR (équation)
4 : MULTIPIPELINE A VECTEUR AVEC FRACTIONNAIRES (équation)
10 : MULTIPIPELINE (LOCEFF)
5 : PROPAGATION PURE (équation)
6 : PROPAGATION LIMITEE (équation)
7 : SUIVI DE CONTOUR (BRESENHAM 8 paramètres)
8 : SUIVI DE CONTOUR (BRESENHAM 5 paramètres)
9 : SUIVI DE CONTOUR (DDA)

Dans un premier temps, nous donnons des valeurs permettant de comparer l'évolution du réseau au cours du tracé d'un segment ((Xa,Ya),(Xb,Yb)).

	SIMD	MULTI-PIPELINE	PROPAGATION PURE		PROPAGATION LIMITEE		SUIVI DE CONTOUR
			4voisins	8voisins	4voisins	8voisins	8voisins
Cellules actives au temps (t)	NxN	Δy	1 à 2(N-1)	1 à 4(N-1)	1 à INF(Δx,Δy)	1 à 2*INF(Δx,Δy)	1
Nombre d'étapes successives	1	N	N	N/2	Δx+Δy	SUP(Δx,Δy)	SUP(Δx,Δy)
Cellules activées	NxN	ΔyxN	NxN		Δx*Δy SUP(Δx,Δy)		
Communication hôte vers cellules	NxN Câblées	Δy Câblées	1 Câblée		1 Protocole (double)		1 Protocole
Communication entre cellules	0	De droite à gauche câblées	(*) (N²-2N) : 1 (2N) : 2 1 : 4 Câblées	(*) (N²-4N) : 1 (4N) : 2 1 : 8 Câblées	Multirectionnelles (diffusion) Câblées		Unidirect. Logiques
Communication cellules vers hôte (contrôle)	0	N Câblées	4N Câblées		Time Out Test d'activité		Protocole

N : Le réseau est de taille NXN; Δx=ABS(Xa,Xa); Δy=ABS(Ya,Yb)

(*) : 1: unidirectionnelle; 2 : bidirectionnelle ...

Figure 33 : TABLEAU DES COMPORTEMENTS SUR LE RESEAU

En complément du tableau précédent, voici celui des calculs nécessaires au tracé d'un segment, établi en fonction des algorithmes proposés en annexe (ANNEXE A).

	SIMD	MULTIPIPELINE				PROPAGATION PURE	PROPAGATION LIMITEE	SUIVI DE CONTOUR		
N° de référence des algos.	1	2	3	(*) 4	(*) 10	5	6	7	8	(*) 9
Nombre de calculs du hôte AVANT envoi	5T 3D 3+ 2 *	5T 3D 3+ 2 *	5T 3D 4+ 3 *	7T 4D 6+ 6 *	3T 4D 4+ 2 *	5T 4D 5+ 3 *	5T 3D 2+	1T 5D 2+	3T 2+	2T 2D 2+ 2 *
ENTRE deux envois	0	0	+	2+	1T 2D 4+	0	0	0		
Nombre d'envois	NxN	Δy				1	1	1		
Nombre de paramètres	8	6	5	5	2	8	8	8	5	6
Nombre de calculs dans la cellule AVANT transmission	5T 1D 2+ 2 *	+	+	2+		+	5T +	2T 3+	3T 2D 3+	2T 2D 4+
APRES transmission	0	3T 1D 1+ 1 *	3T 1D	4T 2D	2T	5T 1D	1T	0		

(*) : Opérations sur des fractionnaires

T: test; D: décalage; +: addition; *: multiplication;

Figure 34 : TABLEAU DES CALCULS POUR UN SEGMENT

En faisant quelques hypothèses complémentaires, nous pouvons déduire des deux tableaux précédents le temps requis pour la visualisation d'un segment. Nous distinguons au cours de cette évaluation le temps d'activation du hôte (T_{hote}), et le temps réseau nécessaire ($T_{réseau}$).

SIMD :

1 $T_{hote} = (11 + 2*Nb) * Op_h + Th_{(NxN)}(8)$
$T_{réseau} = (8 + 2*Nb) * Op_c$

MULTIPIPELINE :

2 $T_{hote} = (11 + 2*Nb) * Op_h + Th_{(\Delta y)}(6)$
$T_{réseau} = N * (Op_c + Tc_{(1)}(6)) + (5 + Nb) * Op_c$

3 $T_{hote} = (12 + 3*Nb + \Delta y) * Op_h + Th_{(\Delta y)}(5)$
$T_{réseau} = N * (Op_c + Tc_{(1)}(5)) + 4*Op_c$

4 $T_{hote} = (17 + 6*Nb + 2*\Delta y) * Op_h + Th_{(\Delta y)}(5)$
$T_{réseau} = N * (2*Op_c + Tc_{(1)}(5)) + 6*Op_c$

10 $T_{hote} = (11 + 2*Nb) * Op_h + \Delta y * (SUP(7*Op_h, Th_{(1)}(2)))$
$T_{réseau} = N * Tc_{(1)}(2) + 2*Op_c$

PROPAGATION PURE :

5 $T_{hote} = (14 + 3*Nb) * Op_h + Th_{(1)}(8)$
$T_{réseau} = N * (Op_c + Tc_{(1)}(8)) + 6*Op_c$ (pour 4 voisins)
$T_{réseau} = N/2 * (Op_c + Tc_{(1)}(8)) + 6*Op_c$ (pour 8 voisins)

PROPAGATION LIMITEE :

8 $T_{hote} = 10*Op_h + Th_{(1)}(8)$
$T_{réseau} = Pt_r(8) + (SUP(\Delta x, \Delta y)) * (Tc_{(8)}(8) + 6*Op_c) + Op_c$
(+ Test d'activité éventuellement)

SUIVI DE CONTOUR :

7 $T_{hote} = 8*Op_h + Th_{(1)}(8)$
$T_{réseau} = Pt_r(8) + (SUP(\Delta x,\Delta y)) * (Tc_{L(1)}(8) + 5*Op_c) + Pt_s$

8 $T_{hote} = 5*Op_h + Th_{(1)}(5)$
$T_{réseau} = Pt_r(5) + (SUP(\Delta x,\Delta y)) * (Tc_{L(1)}(5) + 8*Op_c) + Pt_s$

9 $T_{hote} = (6 + 2*Nb) * Op_h + Th_{(1)}(6)$
$T_{réseau} = Pt_r(6) + (SUP(\Delta x,\Delta y)) * (Tc_{L(1)}(6) + 8*Op_c) + Pt_s$

N : Taille du réseau
Nb : Nombre de bits pour coder un entier
Op_c : Temps pour une opération élémentaire dans une cellule
Op_h : Temps pour une opération élémentaire dans l'ordinateur hôte
$Pt_r(i)$: Protocole de communication dans le réseau pour i paramètres entiers
Pt_s : Protocole de sortie cellule vers hôte (pour le contrôle)
$Th_{(i)}(j)$: Transmission directe du hôte vers i cellules, d'une commande à j paramètres
$Tc_{(i)}(j)$: Transmission directe d'une cellule vers i cellules, d'une commande à j paramètres
$Tc_{L(i)}(j)$: Transmission logique d'une cellule vers i cellules, d'une commande à j paramètres

Attribuons des valeurs aux paramètres :

N = 1000 cellules
Nb = 10 bits
Op_c = 20 ns
Op_h = 100 ns
$Pt_r(i)$ = N/2 * $Tc_{(j)}(i)$ = i * 10000 ns
Pt_s = 10000 ns
$Th_{(i)}(j)$ = j * 100 ns
$Tc_{(i)}(j)$ = j * 20 ns
$Tc_{L(i)}(j)$ = j * 50 ns

Ces valeurs correspondent à une machine hôte classique.

Pour $\Delta x,\Delta y$ nous étudions deux éventalités : $\Delta x,\Delta y$ = N/50 = 20 pixels
et $\Delta x,\Delta y$ = N/3 = 300 pixels

SIMD :

1

Nous ne pouvons pas évaluer cette solution sans faire d'hypothèse sur le mode de distribution des commandes.

Nous proposons trois approches représentatives des différentes options :

1) Accès simultané à toutes les cellules

=> $Th_{(NXN)}(8) = 8*100 = 800$ ns

=> $T_{hote} = (11 + 2*10) * 100 + 800 = 3100 + 800 = 3{,}9$ µs

2) Accès par un arbre binaire

=> $Th_{(NXN)}(8) = Th_{(1)}(8) + \log_2(N) * Tc_{(2)}(8) = 8*100 + 10*8*20 = 2{,}4$ µs

=> $T_{hote} = (11 + 2*10) * 100 + 2400 = 3100 + 2400 = 5{,}5$ µs

3) Accès par un multipipeline

=> $Th_{(NXN)}(8) = Th_{(1)}(8) + N * Tc_{(2)}(8) = 8*100 + 1000*8*20 = 160{,}8$ µs

=> $T_{hote} = (11 + 2*10) * 100 + 160000 = 3100 + 160800 = 164$ µs

Le temps réseau est lui identique pour les trois solutions.

=> $T_{réseau} = (8 + 2*10) * 20 = 560$ ns

Dès maintenant, nous pouvons nous apercevoir qu'une machine hôte classique est incapable d'alimenter correctement un tel réseau, quel que soit par ailleurs le mode de diffusion des commandes choisi.

MULTIPIPELINE :

2

=> $T_{hote} = (11 + 2*10) * 100 + 6*100 = 3{,}7$ µs

=> $T_{réseau} = 1000 * (20 + 6*20) + (5 + 10) * 20 = 140{,}3$ µs

3

$\Rightarrow T_{hote} = (12 + 3*10 + \Delta y) * 100 + 5*100 = \Delta y * 100 + 4700$

$\Delta y = 20 \Rightarrow T_{hote} = 6,7\ \mu s$
$\Delta y = 300 \Rightarrow T_{hote} = 34,7\ \mu s$

$\Rightarrow T_{réseau} = 1000 * (20 + 5*20) + 4*20 = 120\ \mu s$

4

$\Rightarrow T_{hote} = (17 + 6*10 + 2*\Delta y) * 100 + 5*100 = \Delta y * 200 + 8200$

$\Delta y = 20 \Rightarrow T_{hote} = 12,2\ \mu s$
$\Delta y = 300 \Rightarrow T_{hote} = 68,2\ \mu s$

$\Rightarrow T_{réseau} = 1000 * (2*20 + 5*20) + 6*20 = 140\ \mu s$

10

$\Rightarrow T_{hote} = (11 + 2*10) * 20 + \Delta y * SUP(7*100,2*100) = \Delta y * 700 + 620$

$\Delta y = 20 \Rightarrow T_{hote} = 14,6\ \mu s$
$\Delta y = 300 \Rightarrow T_{hote} = 210\ \mu s$

$\Rightarrow T_{réseau} = 1000 * 2 * 20 + 2*20 = 40\ \mu s$

PROPAGATION PURE :

5

$\Rightarrow T_{hote} = (14 + 3*10) * 100 + 8*100 = 5,2\ \mu s$

$\Rightarrow T_{réseau} = 1000 * (20 + 8*20) + 6*20 = 180\ \mu s$ (pour 4 voisins)
$\Rightarrow T_{réseau} = 500 * (20 + 8*20) + 6*20 = 90\ \mu s$ (pour 8 voisins)

PROPAGATION LIMITEE :

6

$\Rightarrow T_{hote} = 10*100 + 8*100 = 1,8\ \mu s$

$\Rightarrow T_{réseau} = 8*10000 + \Delta xy * (8*20 + 6*20) = 80000 + \Delta xy * 280$

$\Delta y = 20 \Rightarrow T_{réseau} = 85,6\ \mu s$
$\Delta y = 300 \Rightarrow T_{réseau} = 164\ \mu s$

SUIVI DE CONTOUR :

7

$\Rightarrow T_{hote} = 10*100 + 8*100 = 1{,}8\ \mu s$

$\Rightarrow T_{réseau} = 8*10000 + \Delta xy * (8*50 + 5*20) + 10000 = 90000 + \Delta xy * 500$

$\Delta xy = 20 \Rightarrow T_{réseau} = 100\ \mu s$
$\Delta xy = 300 \Rightarrow T_{réseau} = 240\ \mu s$

8

$\Rightarrow T_{hote} = 5*100 + 5*100 = 1\ \mu s$

$\Rightarrow T_{réseau} = 5*10000 + \Delta xy * (5*50 + 8*20) + 10000 = 60000 + \Delta xy * 410$

$\Delta xy = 20 \Rightarrow T_{réseau} = 68{,}2\ \mu s$
$\Delta xy = 300 \Rightarrow T_{réseau} = 183\ \mu s$

9

$\Rightarrow T_{hote} = (6 + 2*10) *100 + 6*100 = 3{,}2\ \mu s$

$\Rightarrow T_{réseau} = 6*10000 + \Delta xy * (6*50 + 8*20) + 10000 = 70000 + \Delta xy * 460$

$\Delta xy = 20 \Rightarrow T_{réseau} = 79{,}2\ \mu s$
$\Delta xy = 300 \Rightarrow T_{réseau} = 208\ \mu s$

Dans la plupart des solutions ont s'aperçoit que c'est l'ordinateur hôte qui pêche par manque de performances.

$T_{segment} = T_{hote} + T_{réseau}$

1 1) $T_{segment} = 3900 + 560 = 4{,}4\ \mu s$
2) $T_{segment} = 5500 + 560 = 6\ \mu s$
3) $T_{segment} = 164000 + 560 = 164{,}5\ \mu s$

2 $T_{segment} = 144\ \mu s$

3 $\Delta y = 20 \Rightarrow T_{segment} = 126{,}7\ \mu s$
$\Delta y = 300 \Rightarrow T_{segment} = 154{,}7\ \mu s$

4 $\Delta y = 20 \Rightarrow T_{segment} = 152{,}2\ \mu s$
$\Delta y = 300 \Rightarrow T_{segment} = 208{,}2\ \mu s$

10 $\Delta y = 20 \Rightarrow T_{segment} = 54{,}6\ \mu s$
$\Delta y = 300 \Rightarrow T_{segment} = 250\ \mu s$

- 5 : 4 voisins => $T_{segment}$ = 185,2 µs
  8 voisins => $T_{segment}$ = 95,2 µs

- 6 : Δxy = 20 => $T_{segment}$ = 87,4 µs
  Δxy = 300 => $T_{segment}$ = 165,8 µs

- 7 : Δxy = 20 => $T_{segment}$ = 101,8 µs
  Δxy = 300 => $T_{segment}$ = 241,8 µs

- 8 : Δxy = 20 => $T_{segment}$ = 69,2 µs
  Δxy = 300 => $T_{segment}$ = 184 µs

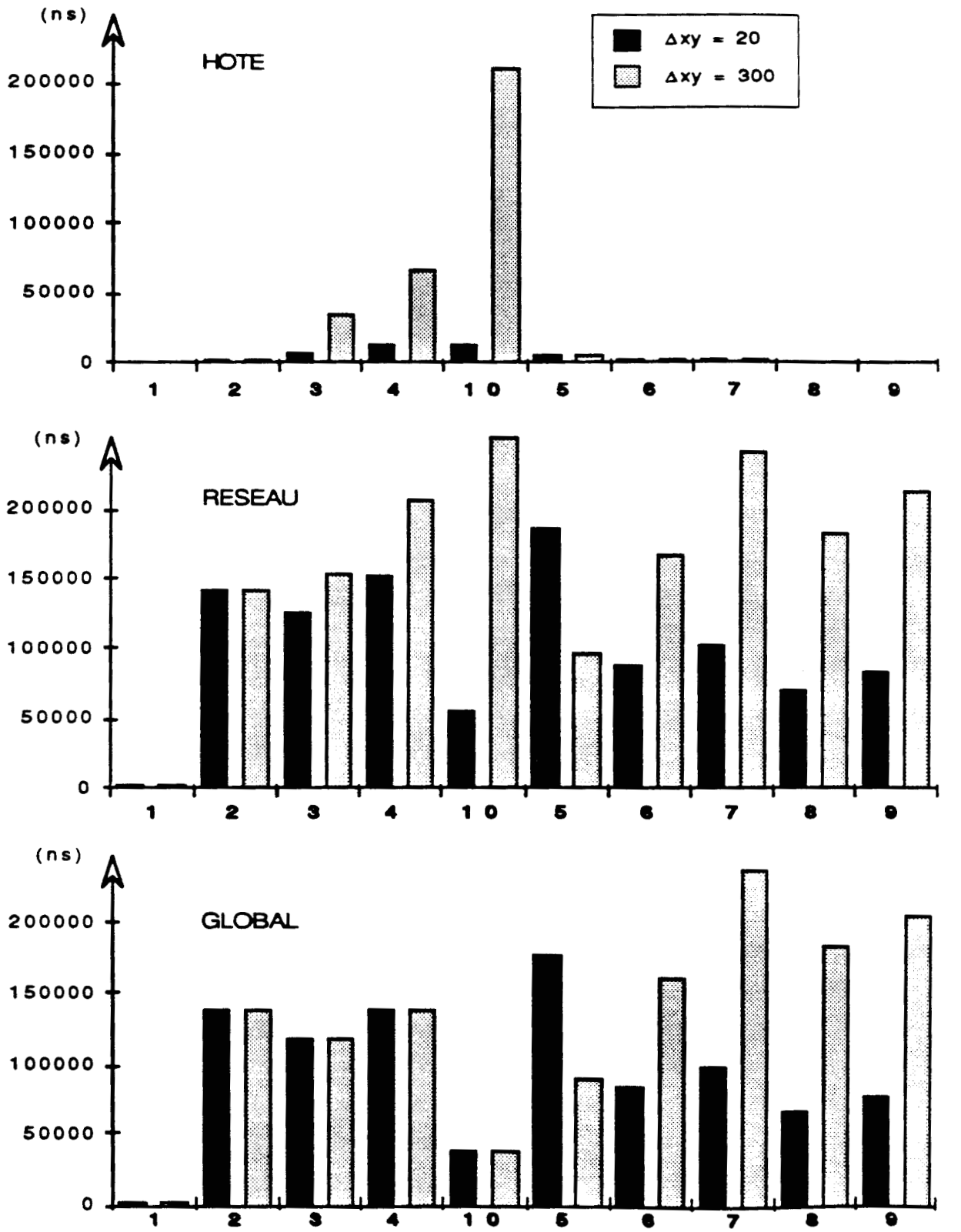
- 9 : Δxy = 20 => $T_{segment}$ = 82,4 µs
  Δxy = 300 => $T_{segment}$ = 211,2 µs

L'évaluation telle quelle des algorithmes conduit inévitablement à choisir la solution SIMD, car le taux de parallélisme objet ne transparaît pas au travers de ces chiffres. Ces chiffres rendent néanmoins partiellement compte du pipeline algorithmique, et totalement du parallélisme algorithmique.

La comparaison à ce niveau n'est pas fondée car elle prend pour hypothèse un cas de figure très particulier (un seul segment à tracer).

Cette comparaison n'est justifiée que par le fait que les valeurs obtenues sont exactes alors que par la suite il s'agira d'approximations.

TEMPS POUR 1 SEGMENT



Nous allons maintenant tenter de comparer le parallélisme objet inhérent à chacune des solutions.

Pour cela, il nous faut faire un certain nombre d'hypothèses sur la composition de l'image. Nous avons choisi de comparer les temps nécessaires à la visualisation d'une scène composée de Ns segments de longueur moyenne $SUP(\Delta x_m, \Delta y_m)$.

Nous avons recours à des approximations assez grossières, peu d'articles traitent des problèmes de distribution des segments dans l'image.

SIMD :

$$T = Ch + Th_A + (Ns-1) * SUP(Ch,Th,Cc) + Cc$$

MULTIPIPELINE :

$$Ne_m = \frac{ENT(\frac{N}{\Delta y_m}) + ENT(\frac{N}{2*\Delta y_m - 1})}{2} = \frac{3}{4} * \frac{N}{\Delta y_m}$$

$$T = Ne_m * Ch + Th + N * CTc + SUP((\frac{Ns}{Ne_m} - 1),0) * SUP(Ne_m * Ch,Th,CTc,Cc) + Cc$$

PROPAGATION PURE :

pour 4 voisins :

$$T = Ch + Th + N * CTc + (Ns-1) * SUP(Ch,Th,CTc,Cc) + Cc$$

pour 8 voisins :

$$T = Ch + Th + N/2 * CTc + (Ns-1) * SUP(Ch,Th,CTc,Cc) + Cc$$

PROPAGATION LIMITEE :

$$Ne_m = \frac{ENT(\frac{N^2}{\Delta y_m{}^2}) + ENT(\frac{N^2}{2*\Delta y_m{}^2 - 1})}{2} = \frac{5}{8} \frac{N^2}{\Delta y_m{}^2}$$

4 voisins :

$$T = Ne_m * Ch + SUP((\frac{Ns}{Ne_m} - 1),0) * (Ne_m * Ch,Th,Tp_r + (\Delta x_m + \Delta y_m) * CTc))$$

8 voisins :

$$T = Ne_m * Ch + SUP((\frac{Ns}{Ne_m} - 1),0) * (Ne_m * Ch,Th,Tp_r + SUP(\Delta x_m,\Delta y_m) * CTc))$$

(+ test d'activité)

SUIVI DE CONTOUR :

$$T = Ne_m * Ch + SUP((\frac{Ns}{Ne_m} - 1),0) * (Ne_m * Ch, Th, Tp_r + \Delta y_m * CTc + Tp_s))$$

$$Ne_m = \frac{N^2}{100}$$

Cc = Calcul dans la cellule
Ch = Calcul dans le hôte
CTc = Calcul dans la cellule avant la transmission + transmission
Th = Transmission du hôte vers le réseau
Th_A = Amorçage du pipeline de distribution (SIMD)
Tp_r = Protocole pour atteindre une cellule
Tp_s = Protocole de sortie du réseau (contrôle)

Ne_m = Nombre d'envois simultanés moyen

Remarque :
Pour l'évaluation du Ne_m du suivi de contour, nous autorisons la possibilité de conflits d'accès entre différents tracés. La valeur que nous cherchons est donc le nombre de tracés qui peuvent être en cours simultanément, sachant que pour chaque tracé il n'y a qu'une cellule à un temps (t). Plus ce nombre est élevé, plus il y a de conflits à gérer, ce qui nuit bien évidemment aux performances. En outre, la gestion des conflits multiples est loin d'être triviale et il faut prendre garde au risque d'inter-blocage. La valeur que nous donnons est plutôt une borne supérieure au delà de laquelle le réseau sera saturé.

En appliquant les hypothèses faites auparavant aux formules précédentes on s'aperçoit rapidement que la cadence de visualisation est directement liée aux performances du hôte.

Un ordinateur séquentiel traditionnel est incapable d'alimenter correctement le réseau, mais une architecture dédiée le peut.

Pour l'évaluation qui suit nous allons, en conservant les hypothèses architecturales sur le réseau, exprimer quelles doivent être les performances du hôte pour obtenir une solution cohérente.

Nous allons d'abord comparer les débits que l'on est susceptible d'obtenir sur les réseaux, en faisant abstraction des performances du hôte.

SIMD

1 Tc=0 & Cc=560 ns ==> Débit = 1 segt/560 ns = 1,7 M segt/secondes

MULTIPIPELINE

2 CTc=140 ns & Cc=320 ns ==> Débit = Ne_m segt/320 ns

Δ = 20 => Ne_m = 32 Débit = 32 segt/320 ns = 96 M segt/secondes
Δ = 300 => Ne_m = 2 Débit = 2 segt/320 ns = 6 M segt/secondes

3 CTc=120 ns & Cc=100 ns ==> Débit = Ne_m segt/120 ns

Δ = 20 => Ne_m = 32 Débit = 32 segt/120 ns = 250 M segt/secondes
Δ = 300 => Ne_m = 2 Débit = 2 segt/120 ns = 16 M segt/secondes

4 CTc=140 ns & Cc=160 ns ==> Débit = Ne_m segt/160 ns

Δ = 20 => Ne_m = 32 Débit = 32 segt/160 ns = 200 M segt/secondes
Δ = 300 => Ne_m = 2 Débit = 2 segt/160 ns = 12,5 M segt/secondes

10 CTc=40 ns & Cc=40 ns ==> Débit = Ne_m segt/40 ns

Δ = 20 => Ne_m = 32 Débit = 32 segt/40 ns = 800 M segt/secondes
Δ = 300 => Ne_m = 2 Débit = 2 segt/40 ns = 50 M segt/secondes

PROPAGATION PURE

5 CTc=180 ns & Cc=140 ns ==> Débit = 1 segt/180 ns = 5,5 M segt/secondes

PROPAGATION LIMITEE

6 80000 + 280 * Δ ns
Δ = 20 => Ne_m = 1500 Débit = 1500 segt/85 µs = 20 M segt/secondes
Δ = 300 => Ne_m = 6 Débit = 6 segt/164 µs = 36000 segt/secondes

SUIVI DE CONTOUR

7 Temps réseau = 80000 + 260 * Δ + 10000 ns, Ne_m = 10000

Δ = 20 Débit = 10000 segt/95 µs = 104 M segt/secondes
Δ = 300 Débit = 10000 segt/174 µs = 56 M segt/secondes

8 Temps réseau = 50000 + 260 * Δ + 10000 ns, Ne_m = 10000

Δ = 20 Débit = 10000 segt/65 µs = 154 M segt/secondes
Δ = 300 Débit = 10000 segt/138 µs = 72 M segt/secondes

9 Temps réseau = 60000 + 280 * Δ + 10000 ns, Ne_m = 10000

Δ = 20 Débit = 10000 segt/75 µs = 133 M segt/secondes
Δ = 300 Débit = 10000 segt/154 µs = 65 M segt/secondes

Il est évident que le meilleur débit du réseau est obtenu pour le multipipeline sur LOCEFF, puisque c'est pour cette solution que le travail des cellules est le moins important.

Il faut maintenant tenter d'évaluer la faisabilité des machines hôtes associées, en fonction du nombre de calculs qu'elles ont a effectuer pour offrir un débit suffisant.

SIMD

1 560 ns pour 32 calculs
55 Mips ==> Débit = 1,7 M segt/secondes

MULTIPIPELINE

2 320 ns pour 32*Ne_m calculs

Δ = 20 => Ne_m = 32 900 calculs
2800 Mips ==> Débit = 96 M segt/secondes

Δ = 300 => Ne_m = 2 62 calculs
190 Mips ==> Débit = 6 M segt/secondes

3 120 ns pour (42+Δ)*Ne_m calculs

Δ = 20 => Ne_m = 32 1860 calculs
15500 Mips ==> Débit = 250 M segt/secondes

Δ = 300 => Ne_m = 2 680 calculs
5500 Mips ==> Débit = 16 M segt/secondes

4 160 ns pour (77+Δ)*Ne_m calculs

Δ = 20 => Ne_m = 32 3104 calculs
19400 Mips ==> Débit = 200 M segt/secondes

Δ = 300 => Ne_m = 2 754 calculs
4700 Mips ==> Débit = 12,5 M segt/secondes

10 40 ns pour (31 + 7*Δ)*Ne_m calculs

Δ = 20 => Ne_m = 32 5472 calculs
137000 Mips ==> Débit = 800 M segt/secondes

Δ = 300 => Ne_m = 2 4200 calculs
105000 Mips ==> Débit = 50 M segt/secondes

PROPAGATION PURE

5 180 ns pour 44 calculs
240 Mips ==> Débit = 5,5 M segt/secondes

PROPAGATION LIMITEE

6 10 * Ne_m calculs
$\Delta = 20$ => $Ne_m = 1500$ 85 µs pour 15000 calculs
200 Mips ==> Débit = 20 M segt/secondes

$\Delta = 300$ => $Ne_m = 6$ 164 µs pour 60 calculs
0,3 Mips ==> Débit = 36000 segt/secondes

SUIVI DE CONTOUR

7 $Ne_m = 10000$
$\Delta = 20$ => 95 µs pour 80000 calculs
840 Mips => Débit = 104 M segt/secondes

$\Delta = 300$ => 174 µs pour 80000 calculs
470 Mips => Débit = 56 M segt/secondes

8 $Ne_m = 10000$
$\Delta = 20$ => 65 µs pour 50000 calculs
770 Mips => Débit = 154 M segt/secondes

$\Delta = 300$ => 138 µs pour 50000 calculs
360 Mips => Débit = 72 M segt/secondes

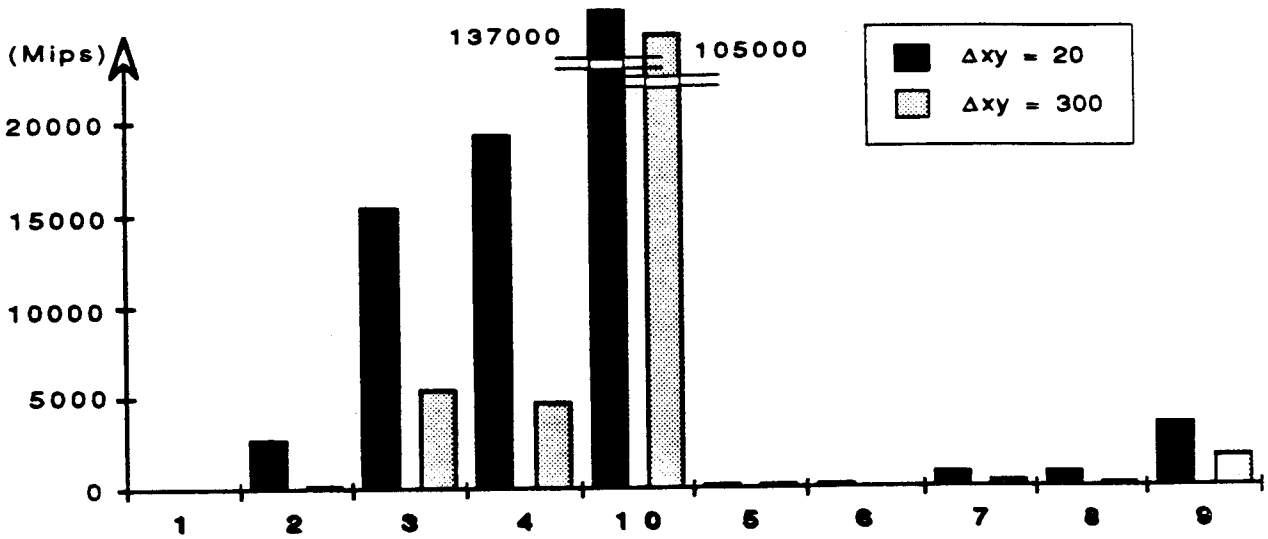
9 $Ne_m = 10000$
$\Delta = 20$ => 75 µs pour 260000 calculs
3460 Mips => Débit = 133 M segt/secondes

$\Delta = 300$ => 154 µs pour 260000 calculs
1680 Mips => Débit = 65 M segt/secondes

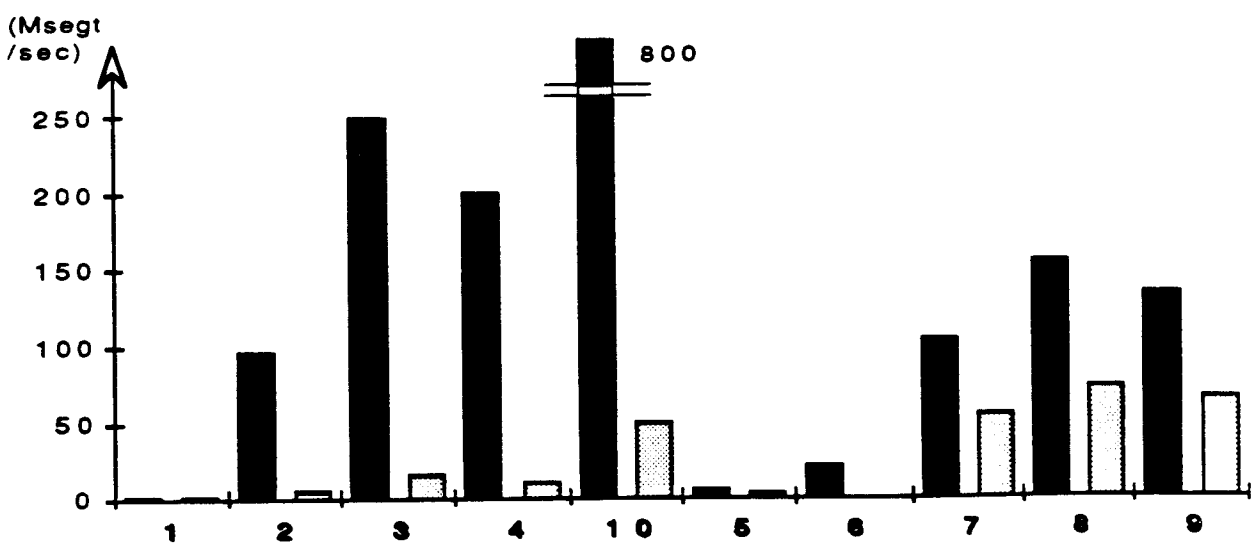
Algorithme N° :	Performances du hôte souhaitable (en Mips)	Performances du réseau attendues (en Msegt/secondes)	Nombre d'instructions par segment
1	55	1,7	32
2	Δy = 20 => 2800 Δy = 300 => 190	96 6	32 32
3	Δy = 20 => 15500 Δy = 300 => 5500	250 16	62 342
4	Δy = 20 => 19400 Δy = 300 => 4700	200 12,5	97 376
10	Δy = 20 => 137000 Δy = 300 => 105000	800 50	171 2100
5	240	5,5	43
6	Δy = 20 => 200 Δy = 300 => 0,3	20 0,03	10 10
7	Δy = 20 => 840 Δy = 300 => 470	104 56	8 8
8	Δy = 20 => 770 Δy = 300 => 360	154 72	5 5
9	Δy = 20 => 3460 Δy = 300 => 1680	133 65	26 26

Figure 52 : Tableau des performances comparées

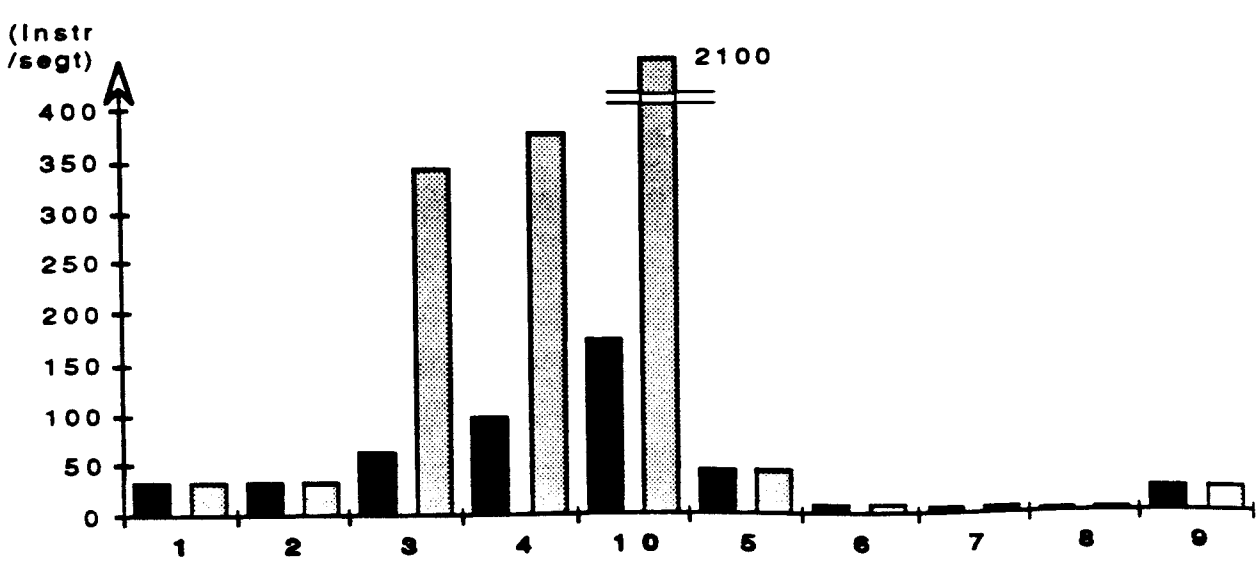
PERFORMANCES DU HOTE



PERFORMANCES DU RESEAU



NOMBRE D'INSTRUCTIONS PAR SEGMENT



L'analyse brutale des chiffres obtenus nous conduirait naturellement à opter pour la solution à suivi de contour 8 qui semble offrir la meilleure adéquation entre la machine hôte et le réseau, tout en ayant de bonnes performances.

Il faut toutefois tempérer notre enthousiasme, en gardant à l'esprit qu'un certain nombre de problèmes non triviaux sont occultés lors de notre évaluation. Citons pour mémoire la gestion des conflits et la réalisation des protocoles de routage. Nous avons de plus fait l'hypothèse d'un taux d'occupation du réseau de 1%, ce qui peut paraître faible mais générera déjà certainement de nombreux conflits. Une réduction de cette occupation diminuerait les performances obtenues dans les mêmes proportions.

Les solutions multipipelines offrent l'avantage d'avoir un comportement totalement prévisible, tout en ayant des performances intéressantes. La solution de LOCEFF 10, si elle possède un excellent débit, sollicite par trop la machine hôte et semble peu réaliste. La proposition 4, qui utilise des fractionnaires, tout en ayant des performances moins bonnes que la 3 réclame plus de calculs du hôte, elle ne sera donc pas retenue. Les solutions 2 et 3 par contre sont assez attrayantes.

Enfin, nous rejetons la propagation limitée, pour cause de résultats insuffisants. La propagation pure a contre elle d'avoir des performances limitées, mais n'est pas exclue pour l'instant, pas plus que la solution SIMD qui est assez peu productive, mais requiert une machine hôte simple.

Remarques :

Plutôt que d'examiner les performances du hôte souhaitable, il est préférable de regarder le nombre d'instructions par segment. En effet, la complexité architecturale n'est pas directement liée au nombre global de calculs de la machine hôte. Certaines solutions se prêtent plus ou moins bien à la parallélisation :

- ◊ La solution SIMD 1 est relativement facile à satisfaire car les exigences du réseau ne sont pas excessives. Il suffit de multiplier les machines de précalcul (dans des proportions assez faibles) et de leur donner accès successivement à l'architecture de distribution.

- ◊ Pour les MULTIPIPELINE 2 3 4 10 il y a deux problèmes : l'un posé par le fait que le nombre de calculs est fonction de la longueur des segments (conduisant à des Nem différents), l'autre est un problème d'accès physique au réseau (pour l'envoi de plusieurs commandes simultanées).
  - Pour les multipipelines à vecteur d'entrée 3 4 l'incrémentation du paramètre dans ledit vecteur peut être faite en parallèle par rapport au reste des calculs, réduisant d'autant la charge du hôte.
  - Pour la solution de LOCEFF 10 il s'agit plus d'envois successifs de commandes pipelines que d'un véritable pipeline, étant donné l'importance des calculs nécessaires entre chaque évaluation de paramètres.

◊ Pour la PROPAGATION PURE 5 la solution est comme en SIMD la multiplication des machines hôtes, mais dans un facteur plus élevé.

◊ La PROPAGATION LIMITEE 6 et le SUIVI DE CONTOUR 7 8 9 conduisent à un petit nombre de calculs mais pour de nombreux segments. La parallélisation est simple étant donné l'indépendance des calculs, mais le taux de parallélisme obtenu est directement lié à la complexité architecturale, et sera par conséquent limité.

Nous étudierons plus en détail ces aspects dans la dernière partie de notre étude.

Même en faisant abstraction des approximations faites au cours des diverses étapes de l'évaluation des performances, les valeurs obtenues ne reflètent pas tous les aspects du comportement des architectures.

Ainsi, par exemple, pour certaines solutions la variation de longueur des segments et leur position n'ont pas d'influence (SIMD, Propagation pure), alors que pour les autres elle est prépondérante.

Cette remarque suggère que le choix de la meilleure architecture dépend de l'utilisation qui en sera faite. Il suffit pour s'en convaincre d'étudier la différence de comportement lors de l'affichage d'une scène composée d'objets à facettes (segments très nombreux (Ns=100000) et très courts (Δ=20)), et d'une image plus classique créée de façon interactive (segments plus longs (Δ=300) et moins nombreux (Ns=10)).

La solution idéale consisterait à comparer le temps nécessaire à l'affichage de scènes types représentatives des différents cas de figure pouvant être rencontrés.

Une machine dédiée à la visualisation de scènes particulières sera toujours meilleure, pour ces scènes, qu'une machine universelle. Notre but est de trouver le meilleur compromis pour réaliser une machine de visualisation polyvalente.

BIBLIOGRAPHIE :

[CHR 80], [DUR 83], [FIE 83], [FRE 61], [HAN 86], [HEG 85], [LIE 88], [LOC 80], [LUC 82], [MAZ 88], [MER 84], [NWS 81], [PEL 85], [PER 88], [REV 88], [ROS 74], [STA 74], [STA 75].

3. LE TRACE DE CERCLES

TABLE DES MATIERES

3. LE TRACE DE CERCLES

3.1. GENERALITES

Il est important de développer de bonnes solutions pour le tracé de cercles, car celles-ci s'étendent de manière évidente au tracé d'arcs de cercles tout aussi fréquemment utilisé [BRE 77], [FPW 87], [PER 88], [PIE 86].

De plus la structure de l'algorithme peut servir de base au développement de solutions pour le tracé des courbes définies par leur équation [HEG 85], [JOR 73].

Pour élaborer un algorithme de tracé de cercles, il existe deux hypothèses de départ distinctes :

- La première basée sur la distance du point par rapport au centre du cercle : Méthode de BRESENHAM.
  Comparaison entre R et $\sqrt{x2+y2}$, pour un point P(x,y), d'un cercle de rayon R centré à l'origine.
- La seconde basée sur l'exploitation des coordonnées polaires des points du cercles : D.D.A.

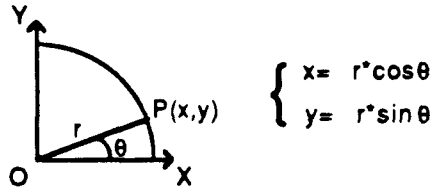


Figure 1 : Equation polaire

3.2. LES ALGORITHMES SEQUENTIELS

3.2.1. ALGORITHME DE BRESENHAM

De même que pour le tracé de segments, pour lequel de nombreux algorithmes ont été développés sur des bases identiques, il existe une multitude d'algorithmes de tracé de cercles reprennant les principes de celui proposé par BRESENHAM.

Cependant, à la différence du segment, les variantes qui en sont issues conduisent parfois à des représentations du cercle différentes.

L'algorithme proposé par BRESENHAM tire profit de certaines particularités du cercle :

- Le cercle est une figure isotrope par rapport à son centre : quatre axes de symétrie sont facilement exploitables pour accélérer la vitesse du tracé.

  Le cercle est donc divisé en octants, le calcul d'un point du tracé permet par la simple application des symétries, de définir les sept points lui correspondant dans les autres octants :

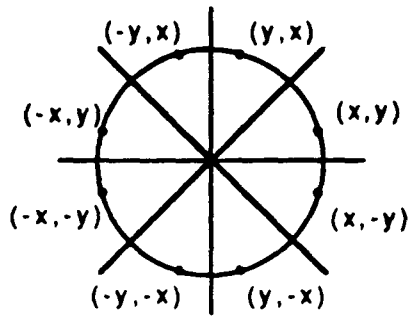


Figure 2 : Symétries entre les octants

- D'autre part, dans chacun de ces octants, le tracé de l'arc de cercle correspondant ne requiert que deux mouvements. Ces couples de directions sont fonction du sens de parcours choisi.

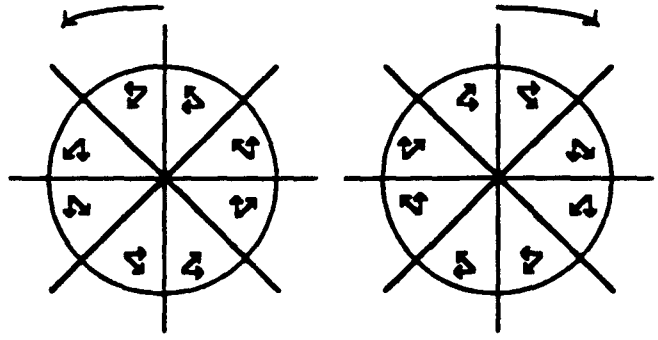


Figure 3 : Mouvements nécessaires

- Concernant le calcul des points du premier octant, quelques remarques à propos de l'équation sont nécessaire :

  L'équation d'un cercle de rayon R centré à l'origine est :

$$\{ (x,y) / x^2 + y^2 = R^2 \}$$

Donc l'erreur E d'un point P(x,y) par rapport au point intersection du cercle et de la droite OP, est telle que :

$$E = x^2 + y^2 - R^2$$

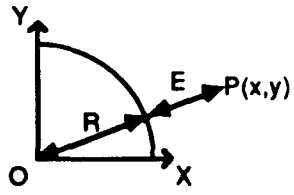


Figure 4 : Erreur

Comme pour le segment une loi de propagation de l'erreur est définie:

Dans le premier octant, pour un parcours dans le sens trigonométrique, les deux seuls mouvements utilisés sont :

$$(x,y) \rightarrow (x,y+1) \quad \& \quad (x,y) \rightarrow (x-1,y+1)$$

$$\text{Or} \quad E(x,y+1) = x^2 + (y+1)^2 - R^2 = E(x,y) + 2*y + 1$$

$$\text{et donc} \quad E(x-1,y+1) = E(x,y) - 2*x + 2*y + 2 = E(x,y+1) - 2*x + 1$$

L'algorithme consiste à choisir, à partir d'un point du cercle, le point suivant introduisant le moins d'erreur possible.

{ cf ANNEXE B : CERCLE SEQUENTIEL 1 }

Remarquons qu'il est possible d'améliorer quelque peu les performances de cet algorithme.

Dans [ILR 83], ILROY remplace le test :

$$\text{Si } (ABS(D_y) >= ABS(D_{xy})) \text{ Alors ... par Si } (D_y >= -D_{xy}) \text{ Alors ...}$$

Ceci est une modification qui n'influe pas sur le tracé obtenu, mais il existe d'autres solutions différant au niveau du test. La plupart du temps elles induisent un tracé légèrement différent. Il faut s'assurer notamment que le tracé obtenu ne présente pas de discontinuité.

Voici succinctement trois possibilités :

- Première proposition [MER 84], [SUE 79], utilisation d'un seuil fixe R : Le nombre de points divergents par rapport au tracé de BRESENHAM est assez important.

- La deuxième solution consiste à utiliser un seuil variable plus simple : 2x. La justification de cette solution se trouve dans [CAR 77], les différences introduites sont minimes.

- Enfin dans [PEL 85], on présente un algorithme utilisant une approximation sur les distances :

$$\text{Distance}((0,0),(x,y)) = \frac{\sqrt{x^2 + y^2 + x + y}}{\sqrt{2}}$$

Ce qui permet d'obtenir un seuil d'affichage S s'exprimant sous la forme : $S = a \times R + b$. Le résultat obtenu empiriquement est en fait S = 0,6 R + 0,28 (ceci pour le deuxième octant).
Le nombre de points divergents par rapport au tracé initial est assez faible (Ex : 124 points pour un cercle de rayon 2048).

Les algorithmes étudiés dans les pages qui précèdent, supposent tous qu'un point possède huit voisins. Mais il est possible de définir des algorithmes pour des points de connectivité quatre, ce qui dans le cadre d'une adaptation cellulaire peut s'avérer particulièrement intéressant.

Malheureusement, ces algorithmes accentuent quelque peu le phénomène de crênelage sur le tracé du cercle.

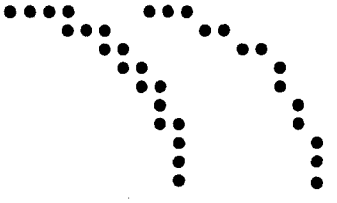


Figure 5 : Connectivité 4 et Connectivité 8

Le principe est basé sur celui retenu dans le cadre de points à huit voisins, simplement plutôt que de comparer les erreurs commises lors des déplacements axiaux et diagonaux, on choisit en fonction des deux mouvements axiaux associés au quadrant à tracer. En effet, le découpage en octant est superflu, et d'ailleurs nous verrons qu'il peut être plus judicieux de diviser le tracé en quadrants pour un réseau, sur lequel l'application des symétries n'est pas aisée.

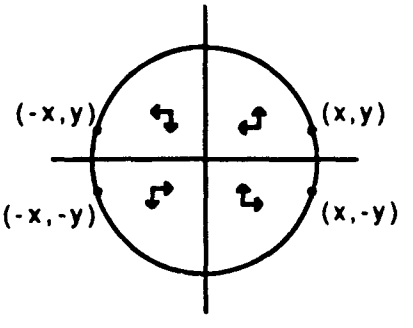


Figure 6 : Symétries entre quadrants

{ cf ANNEXE B : CERCLE SEQUENTIEL 2 }

3.2.2. ALGORITHME DE BISWAS

[BIS 85]

Comme nous l'avons signalé plus haut, l'emploi des symétries pour un algorithme cellulaire n'est pas des plus judicieux, d'où l'intérêt porté aux algorithmes basés sur un découpage en quadrants.

Contrairement à l'algorithme de BRESENHAM qui a recours à trois mouvements pour le tracé d'un quadrant (Ex : ←↖↑ pour le premier), celui-ci n'en nécessite que deux (Ex : ↙↑ pour le premier quadrant), ceci sans altérer la qualité du tracé, contrairement à la limitation à quatre voisins.

{ cf ANNEXE B : CERCLE SEQUENTIEL 3 }

Remarque :
La méthode est toujours incrémentale, mais elle nécessite le calcul de points n'appartenant pas au segment.

3.2.3. ALGORITHME PAR ANALYSE DIFFERENTIELLE (D.D.A.)

[DAN 70] [NWS 81]

Il s'agit d'une méthode basée sur les coordonnées polaires et non plus sur l'évaluation de la distance.

Le principe général est simple : définir une loi permettant d'exprimer les coordonnées d'un point en fonction de celles du point qui précéde. Le calcul de cette relation nécessite quelques approximations.

Le point de départ est néanmoins toujours le même, puisqu'il s'agit de l'équation polaire du cercle.

$$\{ M(x,y) \;/\; x = R*\cos(\theta) \text{ et } y = R*\sin(\theta) \text{ où } \theta=(OX,OM) \}$$

Entre deux points d'un même cercle de rayon R centré à l'origine existe la relation suivante :

$$x_b = x_a * \cos(\theta) + y_a * \sin(\theta)$$
$$y_b = y_a * \cos(\theta) - x_a * \sin(\theta)$$

où $A(x_a,y_a)$ et $B(x_b,y_b)$ appartiennent au cercle et $\theta = (OA,OB)$.

Deux options en sont issues :

- Soit définir un angle unitaire θ, fixer un pas en quelque sorte. Le point suivant est obtenu par l'application au point considéré d'une rotation de θ, puis recherche du point de l'écran approchant au mieux le point calculé.

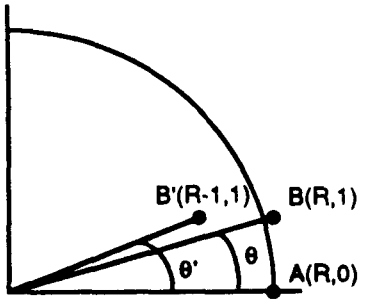


Figure 7 : Choix d'un angle unitaire correct

B(R,1) et B'(R-1,1) sont les deux choix possibles dans ce cas.

Nous avons essayé un algorithme utilisant comme pas l'angle θ, puis l'angle θ', il s'est avéré que θ' est trop grand et conduit donc à un tracé discontinu, alors que θ donne un tracé correct mais avec quelques mouvements nuls.

En fait nous approchons par θ une valeur fixe, celle qu'il a lors du mouvement initial :

$$A(R,0) \text{ et } B(R,1)$$

$$SIN(\theta) = \frac{AB}{OB} = \frac{\sqrt{(R-R)^2+(1-0)^2}}{\sqrt{R^2+1}} = \frac{1}{\sqrt{R^2+1}}$$

$$\text{et } COS(\theta) = \frac{OA}{OB} = \frac{R}{\sqrt{R^2+1}}$$

L'approximation faite sur θ conduit à un algorithme assez rapide car le nombre de mouvements nuls est assez réduit. Cela posera néanmoins quelques problèmes lors de la paralélisation.

{ cf ANNEXE B : CERCLE SEQUENTIEL 4 }

- Dans [MOR 76] une approximation différente est choisie :

$$X_{i+1} = X_i - \frac{Y_i}{R}$$

$$Y_{i+1} = Y_i + \frac{X_{i+1}}{R}$$

{ cf ANNEXE B : CERCLE SEQUENTIEL 5 }

Cet algorithme a pour avantage sur le précédent de ne pas utiliser la fonction racine carrée, et d'être moins gourmand en temps de calcul.

3.2.4. NOTES SUR LES ALGORITHMES SEQUENTIELS

D'abord remarquons que les algorithmes présentés effectuent tous le tracé d'un cercle centré à l'origine. Ceci n'a pas de conséquence grave sur les machines séquentielles puisqu'il suffit, une fois les points du cercle calculés d'appliquer une translation suivant OC, où $C(x_c,y_c)$ est le centre du cercle à tracer.

Cela est obtenu en modifiant les paramètres de la procédure d'affichage:

Afficher (x,y) devient Afficher($x+x_c$,$y+y_c$)

Nous verrons que cela n'est pas aussi évident pour un tracé régi par un réseau cellulaire. Il est en effet préférable de calculer directement les points du cercle correctement positionné, ceci évite des routages complexes au travers du réseau et favorise la répartition des tâches entre les cellules.

Nous avons vu quelques variantes d'algorithmes dont l'objectif principal est de réduire, autant que possible, le temps d'exécution du tracé.

Un autre axe de recherche consiste à améliorer la qualité du tracé, ceci dans le cadre du développement d'applications à but artistique principalement.

Pour ce faire il est nécessaire de définir quelques règles et critères :

- Le polygone approchant le cercle, créé lors du tracé, ne doit pas contenir d'angle droit entrant.

- Aucune des mailles de la grille définissant l'écran ne doit contenir plus de deux points du cercle approché :

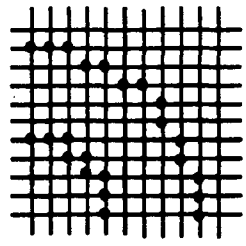


Figure 8 : Tracé incorrect, tracé correct

Ce qui outre les angles droits entrants élimine également les angles droit sortants.
Ces deux critères semblent rejeter les réseaux de connexité quatre, mais il n'en est rien car les procédés d'antialiassage consistent à donner à des points n'appartenant pas au tracé proprement dit, une intensité voisine de la sienne.

- Quels que soient deux points consécutifs du tracé, il existe une maille de la grille qui leur est commune.

- Le tracé doit conserver le caractère symétrique de la figure

Diverses solutions ont été développées, notamment des programmes utilisant des demi-entiers qui fournissent de meilleures approximations [ILR 83].

Le propos de cette étude n'est pas de développer des solutions complexes permettant des tracés quasi parfaits, nous nous contenterons donc des algorithmes présentés précédemment.

3.3. LES ALGORITHMES PARALLELES

Les algorithmes parallèles ont pour objet de construire un cercle de rayon quelconque et centré en un point quelconque, et non plus, comme en séquentiel, centré à l'origine.

Il est difficile de tirer pleinement profit du caractère symétrique de la figure, car il faudrait alors avoir recours à des liaisons entre cellules distantes. Cela demande la conception d'un protocole de communication complexe ayant de nombreux conflits d'accès à résoudre.

3.3.1. ALGORITHMES BASES SUR L'EQUATION DU CERCLE

La solution la plus triviale pour développer un algorithme parallèle consiste, comme pour le segment, à transmettre l'équation à toutes les cellules, puis à évaluer l'erreur commise.

L'équation d'un cercle de rayon R centré en un point $C(x_c,y_c)$ s'exprime sous la forme :

$$\{ (x,y) / (x-x_c)^2 + (y-y_c)^2 = R^2 \}$$

Donc l'erreur en un point P(x,y) est :

$$E(x,y) = (x-x_c)^2 + (y-y_c)^2 - R^2$$

Il est possible d'évaluer le seuil d'affichage S, en remarquant que :

$$E(x+1,y) = (x+1-x_c)^2 + (y-y_c)^2 - R^2 = x^2 - 2*x*x_c + x_c^2 + 2*(x-x_c) + 1 + (y-y_c)^2 - R^2$$

$$\Rightarrow \quad E(x+1,y) = E(x,y) + 2*(x-x_c) + 1$$

de même :

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

$$\Rightarrow S = (2 * (SUP (ABS (x-x_c), ABS (y-y_c))) + 1) DIV 2$$

Remarque :
Le seuil est donc désormais variable, ce qui n'était pas le cas pour le segment. Nous verrons que cela ne va pas sans poser quelques problèmes.

Complétons la loi sur la propagation de l'erreur, pour traiter les mouvements diagonaux :

$$E(x\pm1,y) = E(x,y) \pm 2*(x-x_c) \pm 2*(y-y_c) + 2$$

Développons donc quelques algorithmes appliqués aux architectures suggérées lors de l'étude du tracé de segments.

3.3.1.1. FONCTIONNEMENT SIMD

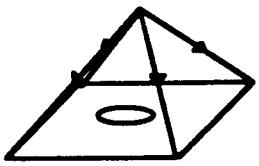


Figure 9 : Diffusion de l'équation

Toutes les cellules reçoivent la même commande de l'ordinateur hôte, le calcul est en grande partie effectué par chacune des cellules.

{ cf ANNEXE B : CERCLE PARALLELE 1 }

3.3.1.2. FONCTIONNEMENT MULTIPIPELINE

Reprenons l'étude du réseau carré accessible par un bord, deux cas se présentent :

- Commande paramétrée unique
- Commande unique avec des paramètres liés à la ligne réceptrice

i) Cas d'une même commande transmise aux cellules du bord du réseau

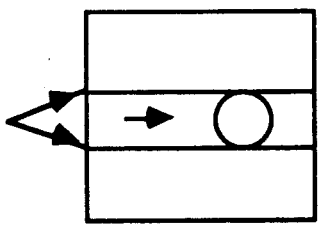


Figure 10 : Multipipeline

Rappel :

Le fonctionnement multipipeline implique un seul type de communication :

Transmission (1,0) de (...)

Or $E(i+1,j) = E(i,j) + 2*(i-x_c) + 1$

où x_c est l'abscisse du centre du cercle

Plutôt que de transmettre x_c de cellules en cellules, c'est ($D_x = x-x_c$) l'abscisse de la cellule relativement au centre, qui sera véhiculée.

Nous ne pouvons envoyer l'erreur totale depuis l'ordinateur-hôte, puisqu'elle est fonction de l'ordonnée de la cellule.

Ce dernier calcule et communique l'erreur initiale en x :

$$E_0 = -R^2 + D_0^2 = -R^2 + x_c^2$$

{ cf ANNEXE B : CERCLE PARALLELE 2 }

Remarque :

Le caractère pipeline est ici plus affirmé que dans le tracé de segments. En effet, la transmission à la cellule voisine a lieu dès réception de la commande, avant qu'aucun calcul ne soit effectué.

ii) Etude du réseau avec des commandes vectorisées

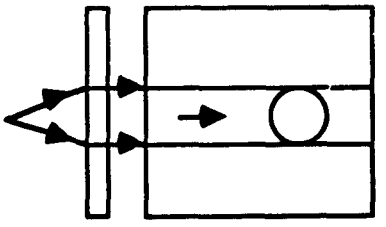


Figure 11 : Multipipeline à vecteur

L'utilisation de ce dispositif permet de réduire la tâche de chacune des cellules.

Avec la commande unique nous ne pouvions pas utiliser le fait que pour toutes les cellules d'une même ligne j, le D_y est constant et égal à $j-y_c$ (où y_c est l'ordonnée du centre).

L'erreur est désormais transmise dans son intégralité et non plus en partie. Cela nécessite l'évaluation de l'erreur initiale :

$$\forall\, j,\quad E(0,j) = -R^2 + x_c^2 + (j-y_c)^2$$

En fait, pour réduire le travail des cellules c'est le double de l'erreur qui est transmis, car c'est lui qui est comparé au seuil d'affichage.

De même, ce n'est pas D_y mais sa valeur absolue qui est utilisée comme paramètre.

{ cf ANNEXE B : CERCLE PARALLELE 3 }

L'algorithme exécuté par la cellule ne requiert plus de multipplication.

3.3.1.3. FONCTIONNEMENT PAR PROPAGATION

Deux cas se présentent :
- propagation à tout le réseau,
- propagation limitée au carré englobant le cercle.

i) Propagation pure

L'erreur initiale au centre du réseau, ainsi que les coordonnées du centre du cercle sont transmises à la cellule centrale. Cela suffit pour l'évaluation incrémentale de l'erreur.

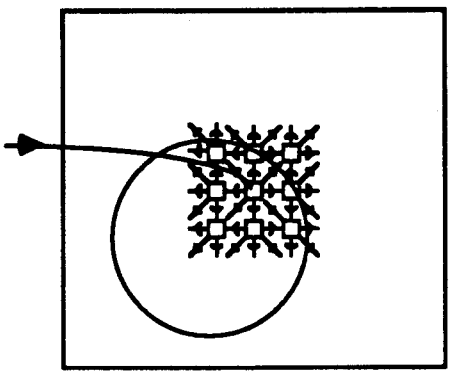


Figure 12 : Propagation pure

{ cf ANNEXE B : CERCLE PARALLELE 4 }

Remarques :

L'algorithme est présenté sans tenir compte des remarques architecturales faites au (2.4.1.2.).
Les modifications à apporter à l'algorithme dans le cadre d'une gestion de la propagation câblée sont triviales.

ii) Propagation limitée

Nous connaissons cinq points du cercle :

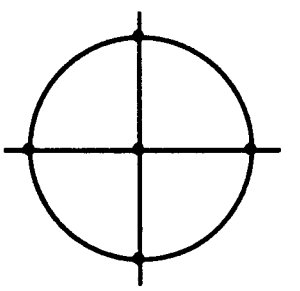


Figure 13 : Points remarquables

En arrêtant la propagation au moment où le cercle est entièrement tracé, sans tenir compte d'éventuelles limitations, les différents schéma de propagation sont :

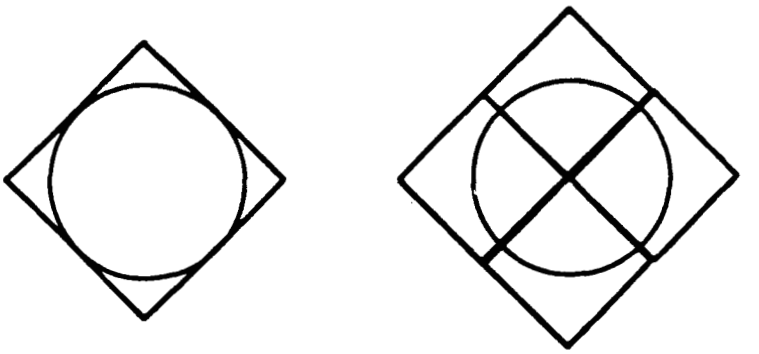


Figure 14 : Propagation limitée (4 voisins)

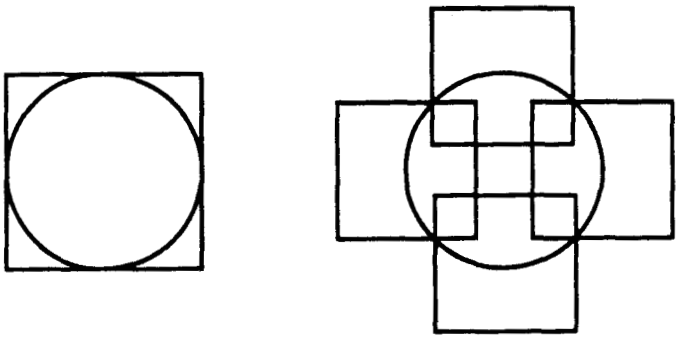


Figure 15 : Propagation limitée (8 voisins)

Dans tous les cas, le temps d'exécution théorique de la solution à quatre points germes est d'environ 70% du temps de la propagation à point unique. Mais la relative complexité des tests d'arrêts à mettre en place, et la multiplication des envois nous conduisent à lui préférer la solution de la propagation à partir du centre.

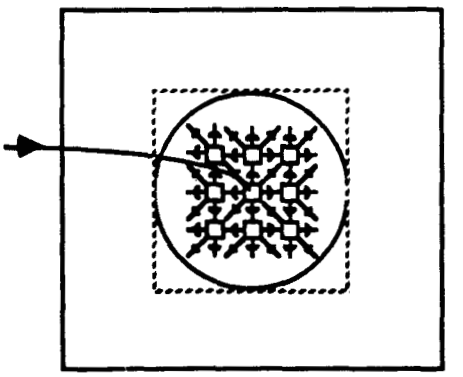


Figure 16 : Propagation limitée (à partir du centre)

En appliquant la loi sur la propagation de l'erreur tout en remarquant que l'erreur initiale, au centre du cercle, est :

$$E_0 = -R^2 + (x_c-x_c)^2 + (y_c-y_c)^2 = -R^2$$

Enfin en utilisant une propagation multidirectionnelle dans huit directions, nous obtenons l'algorithme :

{ cf ANNEXE B : CERCLE PARALLELE 5 }

Remarque :

Le test pour savoir si la cellule n'a pas déjà exécuté le traitement de la commande est difficilement implémentable de façon efficace, à moins d'utiliser une variable spéciale pour le marquage.

Nous proposons donc une autre solution :

Comme nous l'avons vu avec le segment la programmation telle quelle de cet algorithme génère de nombreux conflits lors de son exécution. Nous allons donc mettre à profit les remarques faites pour le segment, pour développer un algorithme sans gestion câblée de la propagation.

Le réseau se subdivise en quatre zones en fonction des directions de propagation requises.

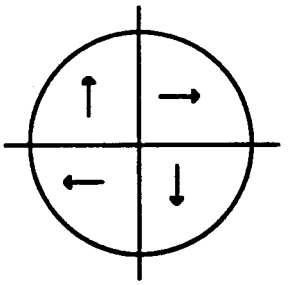


Figure 17 : Propagation limitée dirigée (4 voisins)

Caractérisation des quadrants :

I <=> $x-x_c >= 0$ et $y-y_c >= 0$ => M(1,0)
II <=> $x-x_c <= 0$ et $y-y_c >= 0$ => M(0,1)
III <=> $x-x_c <= 0$ et $y-y_c <= 0$ => M(-1,0)
IV <=> $x-x_c >= 0$ et $y-y_c <= 0$ => M(0,-1)

où $M(d_x,d_y)$ est le mouvement correspondant à un déplacement en x de d_x, et en y de d_y.

Grâce à cette remarque nous développons un algorithme bien plus performant que le précédent, car sans conflit d'accès.

{ cf ANNEXE B : CERCLE PARALLELE 6 }

Le principal reproche à faire à cet algorithme est qu'il nécessite quatre double-tests à chaque exécution pour reconnaître le quadrant, mais il a pour lui sa simplicité et sa concision.

Il existe une variante de cet algorithme ne nécessitant que deux tests par exécution, ses performances sont meilleures mais il est plus long et plus complexe:

Il faut définir une loi pour choisir la ou les directions de propagation en fonction des coordonnées relatives de la cellules par rapport à la cellule initiale.

Rappel :
SGN() : fonction qui prend ses valeurs parmi (-1,0,1) en fonction du signe de l'argument.

Notations :
S_x = SGN $(x-x_c)$ & S_y = SGN $(y-y_c)$ où (x_c,y_c) est le centre du cercle,
M(dx,dy) = Mouvement de déplacement en x égal à dx, et en y égal à dy. dx et dy prennent leurs valeurs dans (-1,0,1).

Etude des différents cas possibles :

- $S_x=S_y=0$ <=> cellule initiale

  propagation dans les quatre directions :

  ⇒ M(1,0), M(0,1), M(-1,0), M(0,-1)

- $S_x=0$ <=> cellule de l'axe vertical

  $S_y=-1$ => M(-1,0) et M(0,-1)
  $S_y=1$ => M(1,0) et M(0,1)

  ⇒ $M(S_y,0)$ et $M(0,S_y)$

- $S_y=0$ <=> cellule de l'axe horizontal

  $S_x=-1$ => M(0,1) et M(-1,0)
  $S_x=1$ => M(1,0) et M(0,-1)

  ⇒ $M(0,-S_x)$ et $M(S_x,0)$

- $S_x<>0$ et $S_y<>0$

  $S_x=1$ & $S_y=1$ ⇒ M(1,0)
  $S_x=1$ & $S_y=-1$ ⇒ M(0,-1)
  $S_x=-1$ & $S_y=-1$ ⇒ M(-1,0)
  $S_x=-1$ & $S_y=1$ ⇒ M(0,1)

  ⇒ $M(SGN(S_x+S_y),SGN(S_y-S_x))$

Cela simplifie donc l'algorithme.

{ cf ANNEXE B : CERCLE PARALLELE 7 }

Comme l'étude architecturale l'a montré, une subdivision de l'espace en octants est également envisageable.

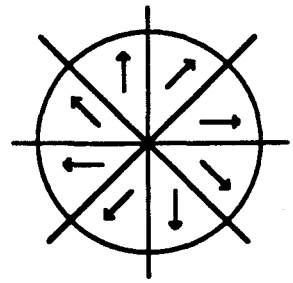


Figure 18 : Propagation limitée dirigée (8 voisins)

Pour identifier la direction associée à une cellule, il semble préférable d'inclure dans la commande un indicateur. De plus cela permet d'illustrer une méthode qui aurait pu être retenue pour la propagation quadri-directionnelle.

Les directions sont codées comme suit :

EST <-> 0, NORD-EST <-> 1, NORD <-> 2, ..., SUD-EST <-> 7

et dans le cas d'une transmission bidirectionnelle on ajoute 8 au code pour la distinguer :

code 9 <=> directions NORD-EST et NORD

Enfin, la cellule initiale reçoit de l'ordinateur un code différent (16) pour aiguiller vers un traitement particulier.

{ cf ANNEXE B : CERCLE PARALLELE 8 }

Les autres solutions pour traiter le cas de la cellule initiale sont :

- Envoyer plusieurs commandes avec des paramètres différents :
  - propagation à 4 voisins => 2 commandes

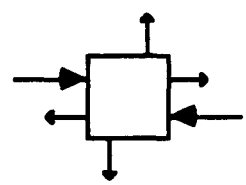


Figure 19 : Envoi double

• propagation à 8 voisins => 4 commandes

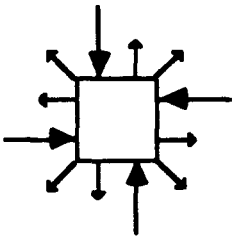


Figure 20 : Envoi quadruple

Cela présente l'inconvénient d'encombrer un peu plus le réseau.

- Définir une commande spéciale pour la première cellule. Cette solution facile à implémenter n'est pas présentée car elle réduit l'espace mémoire de chaque cellule puisque celles-ci doivent alors être capable d'interpréter une commande supplémentaire.

Remarque importante :
Les trois derniers algorithmes proposés s'adaptent sans problèmes à une propagation à partir du centre du réseau. Il suffit pour cela de modifier la valeur de l'erreur transmise à la cellule initiale. Cela supprime le test de limitation au carré englobant, en contrepartie la diffusion est totale.

3.3.2. NOTES SUR LES ALGORITHMES INCREMENTAUX PARALLELISES

Ces algorithmes sont, de par leur structure, à ranger dans la classe des algorithmes à propagation par suivi de contour, selon la définition qui en a été donnée lors de l'étude du tracé de segments.

La solution la plus favorable pour paralléliser ces algorithmes consiste à propager la commande à partir de plusieurs points initiaux.

Or, la définition d'un cercle (la donnée du rayon R et du centre $C(x_c,y_c)$) fournit cinq points remarquables :

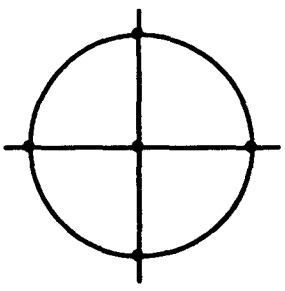


Figure 21 : Points remarquables

Il est donc préférable d'utiliser ces points, mais aucune interdiction n'existe concernant la propagation à partir d'autres points du cercle.

Enfin, les algorithmes séquentiels présentés se limitent au calcul des points d'un octant ou d'un quadrant, et se servent des symétries pour compléter le tracé. De légères modifications sont à apporter pour écrire des algorithmes similaires pour les autres octants, quadrants.

Les algorithmes parallélisés se décomposent donc en plusieurs commandes de tracés d'octants et quadrants.

Plusieurs stratégies sont possibles :

i) Algorithmes découpés en octants :

- Propagation à partir de huit points :

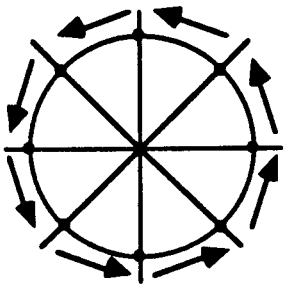


Figure 22 : 8 points

Avantage : Parcours orienté du cercle
Inconvénients : Apparition de quatre nouveaux points qu'il faut calculer sur l'ordinateur hôte.

Cela ne va pas sans poser quelques problèmes d'approximation :

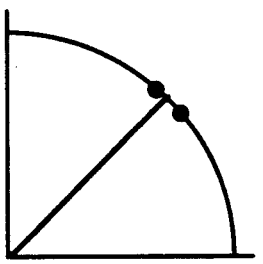


Figure 23 : Points à approcher

- Propagation à partir de sept points :

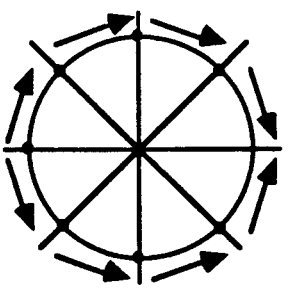


Figure 24 : 7 points

Avantage : Le parcours unilatéral du réseau est intéressant du point de vue architectural, car il limite le nombre de conflits avec d'autres commandes exécutées en parallèle et simplifie la gestion du réseau.
Inconvénients : Le calcul des quatre nouveaux points et les sept routages.

- Propagation à partir de quatre points :

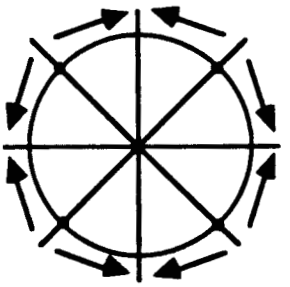


Figure 25 : 4 points

Avantage : Seuls quatre routages sont nécessaires.
Inconvénient : Parcours bilatéral.

ii) Algorithmes découpés en quadrants :

- Propagation à partir de quatre points :

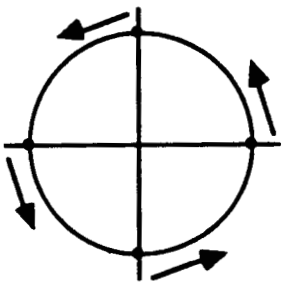


Figure 26 : 4 points

Avantage : Parcours orienté du cercle
Inconvénient : Parcours bilatéral et quatre routages.

- Propagation à partir de trois points :

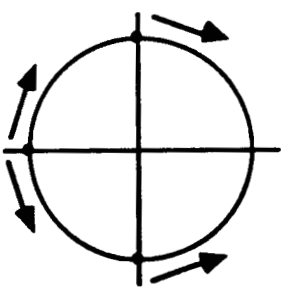


Figure 27 : 3 points

Parcours unilatéral, plus que trois routages.

- Propagation à partir de deux points :

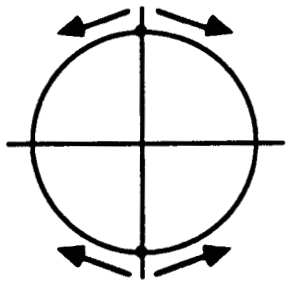


Figure 28 : 2 points

Plus que deux routages mais parcours bilatéral.

iii) Algorithme découpé en deux moitiés :

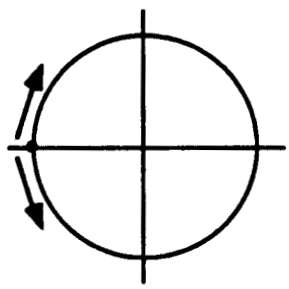


Figure 29 : 1 point

Un seul routage, parcours unilatéral.

Le meilleur compromis doit limiter au maximum :

- Le temps d'exécution du tracé => division en octants,
- Le nombre de commandes à transmettre => 1 point de départ,
- Le précalcul effectué par l'ordinateur hôte.

Malheureusement, le choix dépend fortement des données, de l'image à tracer.

Pour illustrer ce constat, regardons deux exemples :

i) Soit à tracer un image composée de nombreux petits cercles, la préoccupation principale est alors de limiter au maximum le nombre de commandes à émettre, de façon à ne pas trop encombrer le réseau et à ne pas surcharger la machine hôte.
De plus dans ce cas particulier, le rapport de un à quatre existant entre les performances d'un algorithme découpé en octants et celle d'un découpé en moitiés a moins d'influence sur le temps global d'exécution.
Une bonne solution est :

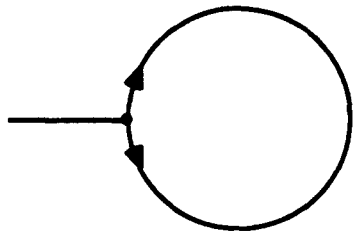


Figure 30 : Propagation à partir d'un point

ii) Si au contraire l'image à tracer comporte des cercles de grande taille, mais peu nombreux, c'est le temps d'exécution d'un tracé qui doit être privilégié, donc on utilise un algorithme découpé en octants.
Par ailleurs l'envoi multiple de commandes est moins pénalisant puisque les cercles sont en nombre réduit.

On s'oriente vers une solution du type :

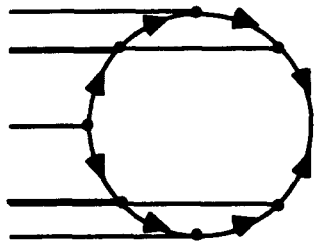


Figure 31 : Propagation à partir de 7 points

Le choix n'est donc pas fixé ici, l'étude portera sur toutes les options présentées.

3.3.3. ALGORITHME DE BRESENHAM CELLULAIRE

Comme l'algorithme proposé par BRESENHAM découpe le cercle en octants, toutes les solutions énoncées dans les pages précédentes sont applicables :

- division en octants,
- division en quadrants,
- division en moitiés.

3.3.3.1. DIVISION EN OCTANTS

Seize algorithmes de tracé d'octants peuvent être rencontrés (8 octants, 2 orientations).

Rappel :

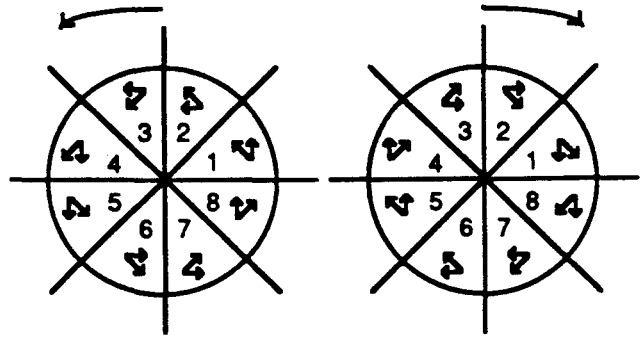


Figure 32 : Numérotation des octants, et mouvements associés

Comme nous le voyons les mouvements associés aux octants sont opposés suivant le sens de parcours choisi. Les algorithmes correspondant sont facilement déductibles les uns des autres.

Dans les programmes présentés ci-après, cette similitude permet des combinaisons de plusieurs mouvements pour réduire la taille et accroître les performances des algorithmes.

Le suivi de contour se propage à partir de plusieurs points, pour implémenter ces envois multiples il y a différentes solutions, en voici trois assez représentatives :

- Définir une commande pour chaque cas (tracé de l'octant1, tracé de l'octant2...) : performant en temps de calcul, mais encombrant l'espace mémoire des cellules.

- Définir une seule commande avec un paramètre supplémentaire pour spécifier le cas à traiter : Le paramètre est interprété par la cellule.

- Une seule commande, la cellule détermine elle-même quel octant, quadrant ou moitié elle doit tracer : Ce calcul supplémentaire grève les performances et augmente la tâche des cellules.

Suivant les algorithmes, nous utiliserons soit la première, soit la seconde proposition.

i) Propagation à partir de sept points :

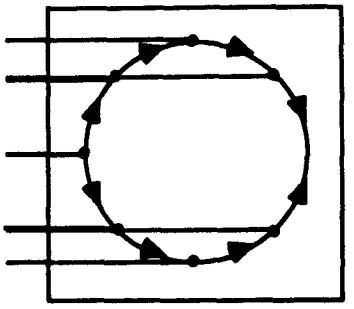


Figure 33 : BRESENHAM à 7 points

Nous utilisons une commande unique avec un paramètre indiquant l'octant à traiter.

Il faut déterminer les quatre points initiaux inconnus à partir de l'équation :

$$x = R\cos(\theta)$$
$$y = R\sin(\theta)$$

avec $\theta = k^* \pi/4$ où $k = 1,3,5,7$

$$\Rightarrow \sin(\theta) = \pm\frac{\sqrt{2}}{2} \text{ et } \cos(\theta) = \pm\frac{\sqrt{2}}{2}$$

Il se pose alors un problème d'approximation puisque les coordonnées obtenues sont réelles. Nous choisirons le point intérieur à l'octant à tracer, l'autre étant tracer par l'octant précédent qui s'arrête sur le test $\pm x = \pm y$.

Les sept points initiaux sont :

$(x_c - R, y_c)$
$(x_c - R_2, y_c + R_2')$
$(x_c - R_2, y_c - R_2)$
$(x_c, y_c + R)$
$(x_c, y_c - R)$
$(x_c + R_2', y_c + R_2')$
$(x_c + R_2', y_c - R_2)$

où (x_c,y_c) est le centre, R le rayon et

$$R_2 = ENT((R^*\frac{\sqrt{2}}{2}) + 0,5) = R_2'$$

Remarque :

- $\frac{\sqrt{2}}{2}$ est remplacé par 0,707 sans modifier le tracé.
- Pour éliminer la double activation d'une cellule commune à deux octants, dans certains cas, on pose :

$$R_2' = ENT((R * \frac{\sqrt{2}}{2}) + 0,99)$$

{ cf ANNEXE B : CERCLE PARALLELE 9 }

ii) Propagation à partir de quatre points :

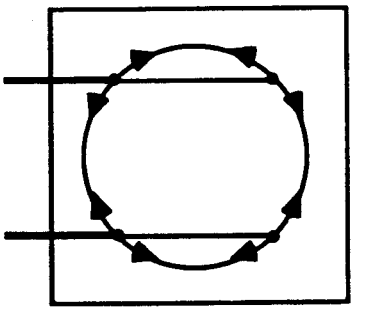


Figure 34 : BRESENHAM à 4 points

Pour implémenter l'algorithme nous définissons quatre commandes distinctes.
Nous reprenons la numérotation classique pour les octants.

L'association deux par deux des octants autorise quelques astuces pour réduire la taille des programmes.

{ cf ANNEXE B : CERCLE PARALLELE 10 }

3.3.3.2. DIVISION EN QUADRANTS

i) Propagation à partir de trois points :

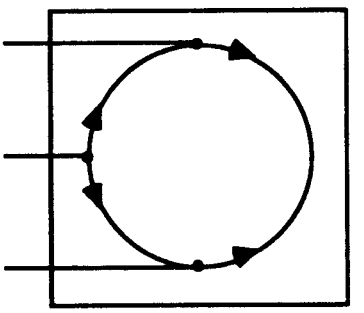


Figure 35 : BRESENHAM à 3 points

Les programmes pour tracer les différents quadrants sont très semblables, une commande unique convenablement paramétrée est donc la meilleure solution pour l'implémentation.

Pour traiter la cellule initiale $(x_c\text{-}R, y_c)$, il y a plusieurs méthodes :

- effectuer, en plus du traitement, un test pour reconnaître s'il s'agit de la cellule initiale et dans l'affirmative dédoubler l'envoi de la commande.

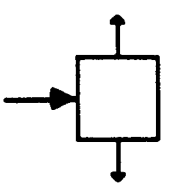


Figure 36 : Envoi dédoublé

Le gros inconvénient de cette solution est que, le programme étant commun à toutes les cellules, le test sera effectué lors de chaque activation.

- envoyer deux commandes, une à la cellule (x_c,y_c) l'autre à la cellule (x_c,y_c-1).

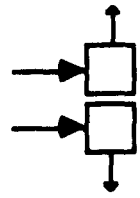


Figure 37 : Double envoi

En effet, les mouvements issus de cette cellule initiale sont nécessairement verticaux et opposés. Le mouvement qui est choisi par l'algorithme est (0,1) puisque :

$$E_x = 0 - 2*R + 1$$
$$E_y = 0 - 2*0 + 1$$
$$E_{xy} = - 2*R + 2$$

$$\Rightarrow ABS(E_y) < ABS(E_x) < ABS(E_{xy})$$

C'est cette dernière solution que nous retenons.

{ cf ANNEXE B : CERCLE PARALLELE 11 }

ii) Propagation à partir de deux points

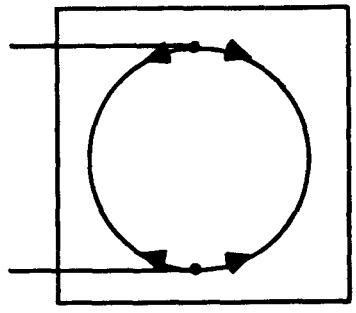


Figure 38 : BRESENHAM à 2 points

Un problème similaire au précédent se pose au sujet des cellules initiales. Pour le résoudre nous adoptons la solution du test supplémentaire, car il est ici trés simple.

Ce test est d'ailleurs effectué après l'envoi *normal*, en outre une remarque sur les déplacements engendrés par les cellules initiales permet de ne faire le test que quand ce dernier est horizontal.

Pour les cellules initiales :

$$E = 0,\ D_x = 0,\ D_y = \pm R$$

$$\Rightarrow\ E_x = E + 1 = 1$$
$$E_y = E \pm 2R + 1 = \pm 2R + 1$$
$$E_{xy} = \pm 2R + 2$$

$$\Rightarrow (ABS(E_x) < ABS(E_y)) \ \&\ (ABS(E_x) < ABS(E_{xy}))$$

=> Le mouvement choisi est donc horizontal. Il ne reste plus alors qu'à effectuer une transmission opposée.

{ cf ANNEXE B : CERCLE PARALLELE 12 }

3.3.3.3. DIVISION EN MOITIES

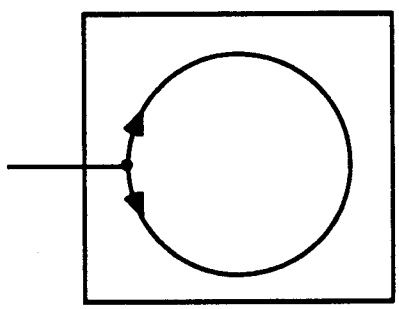


Figure 39 : BRESENHAM à 1 point

Il est intéressant de trouver un algorithme unique pour les deux moitiés, ainsi le tracé ne nécessite plus que l'envoi d'une seule commande.

Le comportement pour la cellule initiale est calqué sur l'exemple précédent, si ce n'est que dans ce cas précis le premier déplacement est vertical.

En effet, si E=0, D_x=-R, D_y=0, ce qui est le cas de la cellule de départ, alors

$$E_x = -2R + 1,\ E_y = 1,\ E_{xy} = -2R + 2\ .$$

Le mouvement choisi sera donc vertical.

{ cf ANNEXE B : CERCLE PARALLELE 13 }

3.3.3.4. CONCLUSION SUR LES ALGORITHMES DE BRESENHAM

Les divers algorithmes proposés ne travaillent que sur les entiers.

De plus, lorsqu'apparaissent des multiplications, celles-ci ne sont là que pour des commodités d'écriture, et peuvent être facilement remplacées. Elles représentent en effet soit des multiplications par 2, soit des changements de signe.

Le câblage de ces algorithmes ne présente aucune difficulté.

Enfin, concernant la parallélisation, on notera que seulement trois ou quatre paramètres, qui plus est entiers, sont transmis de cellules en cellules. Nous verrons que c'est peu comparativement aux autres solutions proposées.

3.3.4. ALGORITHME DE BISWAS PARALLELISE

Rappelons qu'il s'agit d'un algorithme incrémental qui possède la particularité de n'avoir recours qu'à deux mouvements pour le tracé d'un quadrant.

Son adaptation se fait sous forme d'un algorithme à suivi de contour.

Le principe est semblable à celui utilisé pour BRESENHAM : découvrir des similitudes entre les programmes des différents quadrants afin de les regrouper en un seul, et envoyer une commande paramétrée suivant le quadrant.

L'algorithme étant découpé en quadrants, nous choisissons une propagation à partir de trois points :

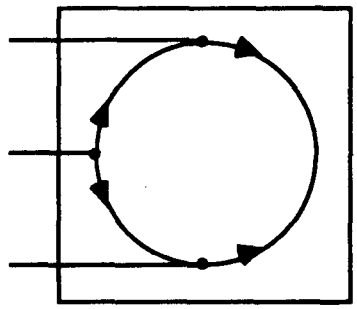


Figure 40 : BISWAS à 3 points

{ cf ANNEXE B : CERCLE PARALLELE 14 }

Remarque :

- D'une part, le nombre de paramètres à transmettre est élevé.
- D'autre part, l'algorithme de par sa conception même, implique l'activation de cellules n'appartenant pas au tracé proprement dit.

3.3.5. ALGORITHME D.D.A. PARALLELISE

Cet algorithme travaille sur des réels et utilise des approximations, mais le traitement, basé sur l'équation polaire, est simple.

L'algorithme en lui-même n'est pas subdivisé, ni en octants, ni en quadrants. Pour le paralléliser il suffit de distribuer la propagation, il s'agit encore d'un algorithme à suivi de contour.

Nous développons le cas d'une propagation à partir d'un point :

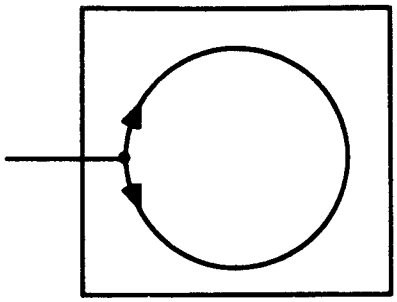


Figure 41 : DDA à deux points

Pour inverser le sens du parcours il suffit de changer le signe de sinus de l'angle élémentaire.

{ cf ANNEXE B : CERCLE PARALLELE 15 }

Remarques :

- Pour traiter la cellule initiale, nous avons recours à un subterfuge, nous transmettons une valeur particulière pour le cosinus.
- Quatre réels transitent de cellules en cellules entraînant des temps de communication plus élevés.

Sur le même schéma nous développons la solution proposée par MORVAN [MOR 76]. Il n'y a plus alors que deux réels et une valeur fixe (le rayon), comme paramètres.

{ cf ANNEXE B : CERCLE PARALLELE 16 }

Pour une parallélisation accrue de ces algorithmes, il faut distribuer la propagation à partir de plusieurs points. Cela nécessite le calcul de nouveaux points initiaux et l'ajustement des tests en conséquence.

3.3.6. ALGORITHME A SYMETRIES

Cette hypothèse tout d'abord rejetée, peut néanmoins être envisagée sur un réseau cellulaire muni d'un protocole de communication performant.

Nous allons étudier le cas d'un réseau avec des communications essentiellement orientées de la gauche vers la droite.

L'algorithme sera dérivé de la méthode de BRESENHAM avec découpage en quadrants. Deux quadrants sont tracés par l'algorithme proprement dit, les autres s'en déduisent par symétrie.

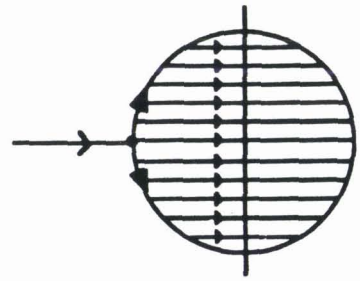


Figure 42 : BRESENHAM à symétries

{ cf ANNEXE B : CERCLE PARALLELE 17 }

Les modifications apportées au programme original sont minimes, et les conflits d'accès sont peu nombreux. Ce ne serait pas le cas pour une utilisation maximale des symétries.

L'emploi des symétries se généralise sans problème à tous les algorithmes de suivi de contour.

Les performances de ces algorithmes sont directement liées à celles du protocole de routage implémenté.

Attention : si l'on tient compte du tracé de disque ces solutions sont trés compétitives.

3.4. CONCLUSION SUR LES ALGORITHMES CELLULAIRES

Des méthodes d'anti-crênelage existent pour chacun des algorithmes présentés, elles sont nombreuses et souvent complexes.
On en trouvera quelques exemples pour les algorithmes traditionnels dans [FIE 86], et également dans [HAN 86]. Pour les algorithmes directement basés sur l'équation, la solution est la même que pour le segment, c'est-à-dire une modification du seuil d'affichage afin d'afficher avec une intensité moindre les points proches du tracé.

3.5. TABLEAUX RECAPITULATIFS

Remarque :

Il n'y a pas de nouvelles observations concernant l'architecture. Ceci est dû à la similitude des solutions avec celles auxquelles a abouti la parallélisation des algorithmes de tracé de segments.
La seule remarque que nous pouvons faire est qu'il n'existe pas d'approche dichotomique pour le tracé cercle. Cette hypothèse n'avait d'ailleurs pas été réellement retenue pour le segment.

Les difficultés rencontrées, lors de l'élaboration des tableaux dressés pour comparer les performances des solutions adoptées pour le tracé de segments, se retrouvent ici.

Les tableaux que nous proposons dans les pages qui suivent sont semblables à ceux présentés pour le segment.

Rappels :

◊ L'hypothèse est faite que les transmissions sont exécutées en parallèle par rapport aux calculs. Cela a pour effet d'introduire des majorants dans les expressions d'évaluation, indiquant si c'est la transmission ou le calcul qui ralentit l'exécution.

◊ On admet que le traitement d'un fractionnaire nécessite deux fois plus de temps que celui d'un entier.

Dans un premier temps, nous donnons des valeurs permettant de comparer l'évolution du réseau au cours du tracé d'un cercle de rayon R.

Rappels : Numérotation des algorithmes

1 : SIMD (équation)
2 : MULTIPIPELINE (équation)
3 : MULTIPIPELINE A VECTEUR (équation)
4,6 : PROPAGATION PURE (équation)
5,7,8 : PROPAGATION LIMITEE (équation)
9 : SUIVI DE CONTOUR (BRESENHAM 7 points)
10 : SUIVI DE CONTOUR (BRESENHAM 4 points)
11 : SUIVI DE CONTOUR (BRESENHAM 3 points)
12 : SUIVI DE CONTOUR (BRESENHAM 2 points)
13 : SUIVI DE CONTOUR (BRESENHAM 1 point)
14 : SUIVI DE CONTOUR (BISWAS 3 points)
15 : SUIVI DE CONTOUR (DDA 1 points)
16 : SUIVI DE CONTOUR (MORVAN 1 points)

	SIMD	MULTI-PIPELINE	PROPAGATION PURE		PROPAGATION LIMITEE		SUIVI DE CONTOUR
			4voisins	8voisins	4voisins	8voisins	8voisins
Cellules actives au temps (t)	NxN	2*R	1 à 2(N-1)	1 à 4(N-1)	1 à 4*(R-1)	1 à 8*(R-1)	NG
Nombre d'étapes successives	1	N	N	N/2	2*(R-1)	R-1	$ENT\left(\frac{2*\Pi*R}{NG}\right)+1$
Cellules activées	NxN	(2R-1)xN	NxN		R*R		6*R
Communication hôte vers cellules	NxN Câblées	(2R-1) Câblées	1 Câblée		1 Protocole		NcG Protocoles
Communication entre cellules	0	Droite à gauche câblées	(*) (N^2-2N) : 1 (2N) : 2 1 : 4 Câblées	(*) (N^2-4N) : 1 (4N) : 2 1 : 8 Câblées	Multirectionnelles Câblées		Unidirect. Logiques
Communication cellules vers hôte (contrôle)	0	N Câblées	4N Câblées		Time Out Test d'activité		Protocole

N : Le réseau est de taille NXN;
NG : Nombre de germes;
NcG : Nombre de cellules germes (1 ou 2 germes par cellule);

(*) : 1: unidirectionnelle; 2 : bidirectionnelle ...

Figure 43 : TABLEAU DES COMPORTEMENTS SUR LE RESEAU

En complément du tableau précédent, voici celui des calculs nécessaires au tracé d'un cercle, établi en fonction des algorithmes proposés en annexe (ANNEXE B).

	SIMD	MULTI		PROPA PURE		PROPA LIMITEE		SUIVI DE CONTOUR				
N° de référence des algos.	1	2	3	4	6	5	7 8	9	1 0	1 1	1 2	1 3
Nombre de calculs du hôte AVANT envoi	*	+ 2 *	2D + 2 *	*		*		3D 3+ 4 *	+	+	+	+
ENTRE deux envois	0	0	D 3+	0		0		0	+	+	+	0
Nombre d'envois	NxN	2xR		1		1		(*) 8 (7)	(*) 8 (4)	(*) 4 (3)	(*) 4 (2)	(*) 2 (1)
Nombre de paramètres	3	3		3		3		4	4	4	3	3
Nombre de calculs dans la cellule AVANT transmission	2T 5D 5+ 2 *	D 3+	2D 3+	3T 3D 3+	3T 4D 4+	7T 3D 3+	6T 4D 6+	2T 4D 6+	4T 7D 4+	6T 5D 8+	4T 8D 8+	6T 4D 7+
APRES transmission	0	2T 5D 3+ *	2T 3D +	2T 5D +		2T 5D +		0	0	0	0	0

(*) : Nombre de germes (NG), entre parenthèses le nombre de cellules germes (NcG).

T: test; D: décalage; +: addition; *: multiplication;

Figure 44 : TABLEAU DES CALCULS POUR UN CERCLE

Remarque :

◊ Pour l'évaluation nous avons remplacé les changements de signe qui était obtenu par des multiplications, par des tests moins gourmands en temps de calcul :

```
X = SGN(A) * B <=> CASE SGN(A) OF
                     0  : X = 0;
                     1  : X = B;
                     -1 : X = -B;
```

◊ Si l'on en était pas convaincu, on remarque que moins il y a de germes, plus le travail des cellules est important, même en tenant compte des quelques modifications apportées entre les différentes variantes. Cela vient s'ajouter au fait qu'il y a plus d'étapes lors du tracé.

Nous dressons maintenant un tableau comparatif des autres solutions proposées pour le suivi de contour :

	1 1	1 4	1 S	1 6	1 8
Nombre de calculs du hôte AVANT envoi	+	3D 5+ *	+	2+ 3 * $\sqrt{...}$	+
ENTRE deux envois	+	+	+	0	0
Nombre d'envois	3	4	1	1	1
Nombre de paramètres	4	6	3	6	5
Nombre de calculs dans la cellule AVANT transmission	6T 5D 8+	8T 3D 5+	6T 4D 7+	3T 10D 8+ 8 *	5T 8D 6 *
APRES transmission	0	0	0	0	0

Figure 45 : Comparaison des solutions à suivi de contour

Remarque :

- ◊ La variante fondée sur MORVAN [16] est meilleure que le DDA [15] mais reste nettement moins bonne que celle de BRESENHAM [13] correspondante.
- ◊ La solution de BISWAS [14] n'apporte rien à l'étude, et possède des performances comparables à celles obtenues avec BRESENHAM [11].

Nous nous contenterons d'étudier les différentes manières d'implanter l'algorithme par suivi de contour développé à partir de la solution de BRESENHAM.

En faisant quelques hypothèses complémentaires, nous pouvons déduire des tableaux précédents le temps requis pour la visualisation d'un cercle. Nous distinguons au cours de cette évaluation le temps d'activation du hôte (T_{hote}), et le temps réseau nécessaire ($T_{réseau}$).

SIMD :

[1] $T_{hote} = Nb * Op_h + Th_{(NxN)}(3)$
$T_{réseau} = (12 + 2*Nb) * Op_c$

MULTIPIPELINE :

[2] $T_{hote} = (1 + 2*Nb) * Op_h + Th_{(2R)}(3)$
$T_{réseau} = N * (4*Op_c + Tc_{(1)}(3)) + (10 + Nb) * Op_c$

[3] $T_{hote} = (3 + 2*Nb + 4*2*R) * Op_h + Th_{(2R)}(3)$
$T_{réseau} = N * (3*Op_c + Tc_{(1)}(3)) + 6*Op_c$

PROPAGATION PURE :

[6] $T_{hote} = Nb * Op_h + Th_{(1)}(3)$
$T_{réseau} = N * (10*Op_c + Tc_{(1)}(3)) + 6*Op_c$ (pour 4 voisins)
$T_{réseau} = N/2 * (11*Op_c + Tc_{(1)}(3)) + 6*Op_c$ (pour 8 voisins)

PROPAGATION LIMITEE :

[6] $T_{hote} = Nb*Op_h + Th_{(1)}(3)$
$T_{réseau} = Pt_r(3) + (2*R-1) * (Tc_{(8)}(3) + 21*Op_c) + 8*Op_c$ (pour 4 voisins)
$T_{réseau} = Pt_r(3) + (R-1) * (Tc_{(8)}(3) + 24*Op_c) + 8*Op_c$ (pour 8 voisins)

(+ Test d'activité éventuellement)

SUIVI DE CONTOUR :

$$N_{étapes} = ENT\left(\frac{ENT(2*\Pi*R)}{NG}\right)$$

0 NG = 8

$T_{hote} = (6 + 4*Nb + 8) * Op_h + Th_{(1)}(4)$

$T_{réseau} = Pt_r(4) + (N_{étapes}) * (Tc_{L(1)}(4) + 12*Op_c) + Pt_s$

1 0 NG = 8

$T_{hote} = (1+8) * Op_h + Th_{(1)}(4)$

$T_{réseau} = Pt_r(4) + (N_{étapes}) * (Tc_{L(1)}(4) + 15*Op_c) + Pt_s$

1 1 NG = 4

$T_{hote} = (1+4) * Op_h + Th_{(1)}(4)$

$T_{réseau} = Pt_r(4) + (N_{étapes}) * (Tc_{L(1)}(4) + 19*Op_c) + Pt_s$

1 2 NG = 4

$T_{hote} = (1+2) * Op_h + Th_{(1)}(3)$

$T_{réseau} = Pt_r(3) + (N_{étapes}) * (Tc_{L(1)}(3) + 20*Op_c) + Pt_s$

1 3 NG = 2

$T_{hote} = Op_h + Th_{(1)}(3)$

$T_{réseau} = Pt_r(3) + (N_{étapes}) * (Tc_{L(1)}(3) + 17*Op_c) + Pt_s$

N : Taille du réseau
Nb : Nombre de bits pour coder un entier
Op_c : Temps pour une opération élémentaire dans une cellule
Op_h : Temps pour une opération élémentaire dans l'ordinateur hôte
$Pt_r(i)$: Protocole de communication dans le réseau pour i paramètres entiers
Pt_s : Protocole de sortie cellule vers hôte (pour le contrôle)
$Th_{(i)}(j)$: Transmission directe du hôte vers i cellules, d'une commande à j paramètres
$Tc_{(i)}(j)$: Transmission directe d'une cellule vers i cellules, d'une commande à j paramètres
$Tc_{L(i)}(j)$: Transmission logique d'une cellule vers i cellules, d'une commande à j paramètres

Attribuons des valeurs aux paramètres :

N = 1000 cellules
Nb = 10 bits
Op_c = 20 ns
Op_h = 100 ns
$Pt_r(i)$ = N/2 * $Tc_{(j)}(i)$ = i * 10000 ns
Pt_s = 10000 ns
$Th_{(i)}(j)$ = j * 100 ns
$Tc_{(i)}(j)$ = j * 20 ns
$Tc_{L(i)}(j)$ = j * 50 ns

Pour R nous étudions deux éventualités : 2*R = N/50 = 20 pixels
et 2*R = N/3 = 300 pixels

Nous conservons l'hypothèse d'une machine hôte traditionnelle, pour l'instant.

Voici les temps nécessaires au tracé d'un cercle :

SIMD :

1

Nous ne pouvons pas évaluer cette solution sans faire d'hypothèse sur le mode de distribution des commandes.

Nous proposons trois approches représentatives des différentes options :

1) Accès simultané à toutes les cellules

=> $Th_{(NXN)}(3) = 3*100 = 300$ ns

=> $T_{hote} = 10 * 100 + 300 = 1{,}3$ µs

2) Accès par un arbre binaire

=> $Th_{(NXN)}(3) = Th_{(1)}(3) + \log_2(N) * Tc_{(2)}(3) = 3*100 + 10*3*20 = 0{,}9$ µs

=> $T_{hote} = 10 * 100 + 900 = 1{,}9$ µs

3) Accès par un multipipeline

=> $Th_{(NXN)}(3) = Th_{(1)}(3) + N * Tc_{(2)}(3) = 3*100 + 1000*3*20 = 60$ µs

=> $T_{hote} = 10 * 100 + 60000 = 61$ µs

Le temps réseau est lui identique pour les trois solutions.

=> $T_{réseau} = (12 + 2*10) * 20 = 640$ ns

MULTIPIPELINE :

2

=> $T_{hote} = (1 + 2*10) * 100 + 3*100 = 2{,}4$ µs

=> $T_{réseau} = 1000 * (4*20 + 3*20) + (10 + 10) * 20 = 140{,}4$ µs

❸

$\Rightarrow T_{hote} = (3 + 2*10 + 2*R*4) * 100 + 3*100 = 8*R * 100 + 2600$

$2*R = 20 \Rightarrow T_{hote} = 10{,}6\ \mu s$
$2*R = 300 \Rightarrow T_{hote} = 120{,}2\ \mu s$

$\Rightarrow T_{réseau} = 1000 * (4*20 + 3*20) + 6*20 = 140\ \mu s$

PROPAGATION PURE :

❻

$\Rightarrow T_{hote} = 10 * 100 + 3*100 = 1{,}3\ \mu s$

$\Rightarrow T_{réseau} = 1000 * (9*20 + 3*20) + 6*20 = 240\ \mu s$ (pour 4 voisins)
$\Rightarrow T_{réseau} = 500 * (10*20 + 8*20) + 3*20 = 130\ \mu s$ (pour 8 voisins)

PROPAGATION LIMITEE :

❺

$\Rightarrow T_{hote} = 10*100 + 3*100 = 1{,}3\ \mu s$

$\Rightarrow T_{réseau} = 3*10000 + (2*R-1) * (3*20 + 13*20) + 8*20 = 30000 + 2*R * 320$

$2*R = 20 \Rightarrow T_{réseau} = 36\ \mu s$
$2*R = 300 \Rightarrow T_{réseau} = 126\ \mu s$

$\Rightarrow T_{réseau} = 3*10000 + (R-1) * (3*20 + 16*20) + 8*20 = 30000 + 2*R * 190$

$2*R = 20 \Rightarrow T_{réseau} = 33{,}8\ \mu s$
$2*R = 300 \Rightarrow T_{réseau} = 87\ \mu s$

SUIVI DE CONTOUR :

❾

$\Rightarrow T_{hote} = (6 + 4*10 + 8) *100 + 4*100 = 5{,}8\ \mu s$

$\Rightarrow T_{réseau} = 4*10000 + N_{étapes} * (4*50 + 12*20) + 10000 = 50000 + N_{étapes} * 440$

$2*R = 20 \Rightarrow N_{étapes} = ENT\left(\frac{\Pi*20}{8}\right) + 1 = 7 \Rightarrow T_{réseau} = 53\ \mu s$

$2*R = 300 \Rightarrow N_{étapes} = ENT\left(\frac{\Pi*300}{8}\right) + 1 = 38 \Rightarrow T_{réseau} = 66{,}7\ \mu s$

1 0

$\Rightarrow T_{hote} = 9 * 100 + 4 * 100 = 1{,}3\ \mu s$

$\Rightarrow T_{réseau} = 4 * 10000 + N_{étapes} * (4 * 50 + 15 * 20) + 10000 = 50000 + N_{étapes} * 500$

$2 * R = 20 \Rightarrow N_{étapes} = ENT(\frac{\Pi * 20}{8}) + 1 = 7 \Rightarrow T_{réseau} = 53{,}5\ \mu s$

$2 * R = 300 \Rightarrow N_{étapes} = ENT(\frac{\Pi * 300}{8}) + 1 = 115 \Rightarrow T_{réseau} = 107{,}5\ \mu s$

1 1

$\Rightarrow T_{hote} = (1+4) * 100 + 4 * 100 = 0{,}9\ \mu s$

$\Rightarrow T_{réseau} = 4 * 10000 + N_{étapes} * (4 * 50 + 19 * 20) + 10000 = 50000 + N_{étapes} * 580$

$2 * R = 20 \Rightarrow N_{étapes} = ENT(\frac{\Pi * 20}{4}) + 1 = 16 \Rightarrow T_{réseau} = 56\ \mu s$

$2 * R = 300 \Rightarrow N_{étapes} = ENT(\frac{\Pi * 300}{4}) + 1 = 233 \Rightarrow T_{réseau} = 181\ \mu s$

1 2

$\Rightarrow T_{hote} = (1+2) * 100 + 3 * 100 = 0{,}6\ \mu s$

$\Rightarrow T_{réseau} = 3 * 10000 + N_{étapes} * (3 * 50 + 20 * 20) + 10000 = 40000 + N_{étapes} * 550$

$2 * R = 20 \Rightarrow N_{étapes} = ENT(\frac{\Pi * 20}{4}) + 1 = 16 \Rightarrow T_{réseau} = 48\ \mu s$

$2 * R = 300 \Rightarrow N_{étapes} = ENT(\frac{\Pi * 300}{4}) + 1 = 233 \Rightarrow T_{réseau} = 168\ \mu s$

1 3

$\Rightarrow T_{hote} = 100 + 3 * 100 = 0{,}4\ \mu s$

$\Rightarrow T_{réseau} = 3 * 10000 + N_{étapes} * (3 * 50 + 17 * 20) + 10000 = 40000 + N_{étapes} * 490$

$2 * R = 20 \Rightarrow N_{étapes} = ENT(\frac{\Pi * 20}{2}) + 1 = 32 \Rightarrow T_{réseau} = 45{,}7\ \mu s$

$2 * R = 300 \Rightarrow N_{étapes} = ENT(\frac{\Pi * 300}{2}) + 1 = 465 \Rightarrow T_{réseau} = 273\ \mu s$

Le temps global pour un tracé de cercle est donc :

$T_{cercle} = T_{hote} + T_{réseau}$

1

1) T_{cercle} =1300 + 640 = 1,9 µs
2) T_{cercle} = 1900 + 640 = 2,5 µs
3) T_{cercle} = 61000 + 640 = 61 µs

2

T_{cercle} = 142,8 µs

3

2*R = 20 => T_{cercle} = 150,6 µs
2*R = 300 => T_{cercle} = 260 µs

6

4 voisins => T_{cercle} = 241,3 µs
8 voisins => T_{cercle} = 131,3 µs

5

4 voisins
2*R = 20 => T_{cercle} = 37,3 µs
2*R = 300 => T_{cercle} = 127,3 µs

8 voisins
2*R = 20 => T_{cercle} = 33,8 µs
2*R = 300 => T_{cercle} = 88 µs

9

2*R = 20 => T_{cercle} = 58,8 µs
2*R = 300 => T_{cercle} = 72,5 µs

10

2*R = 20 => T_{cercle} = 54,8 µs
2*R = 300 => T_{cercle} = 108,8 µs

11

2*R = 20 => T_{cercle} = 56,9 µs
2*R = 300 => T_{cercle} = 181,9 µs

12

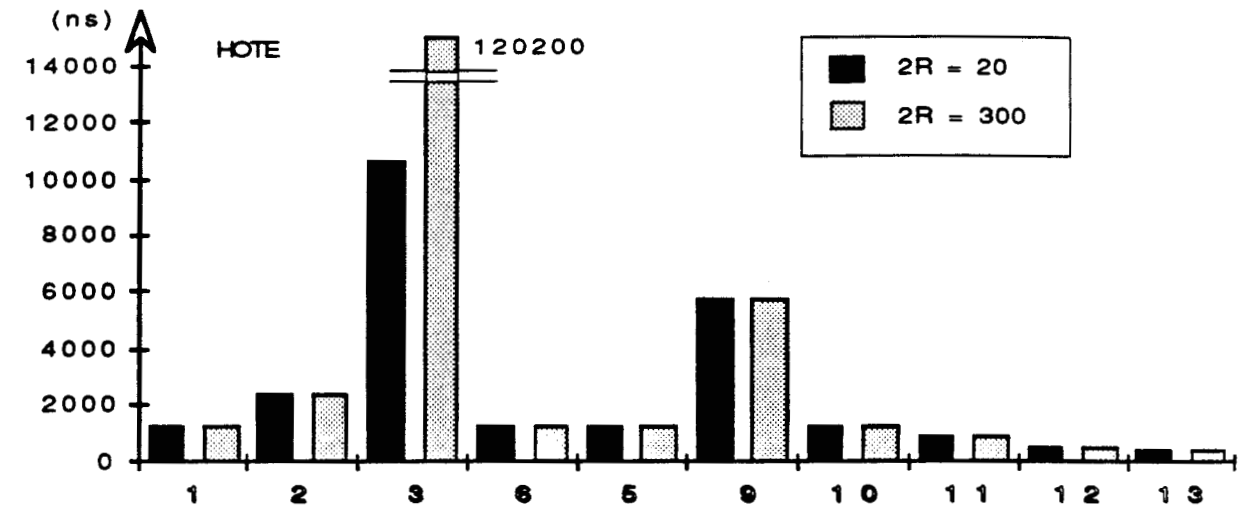
2*R = 20 => T_{cercle} = 48,6 µs
2*R = 300 => T_{cercle} = 168,6 µs

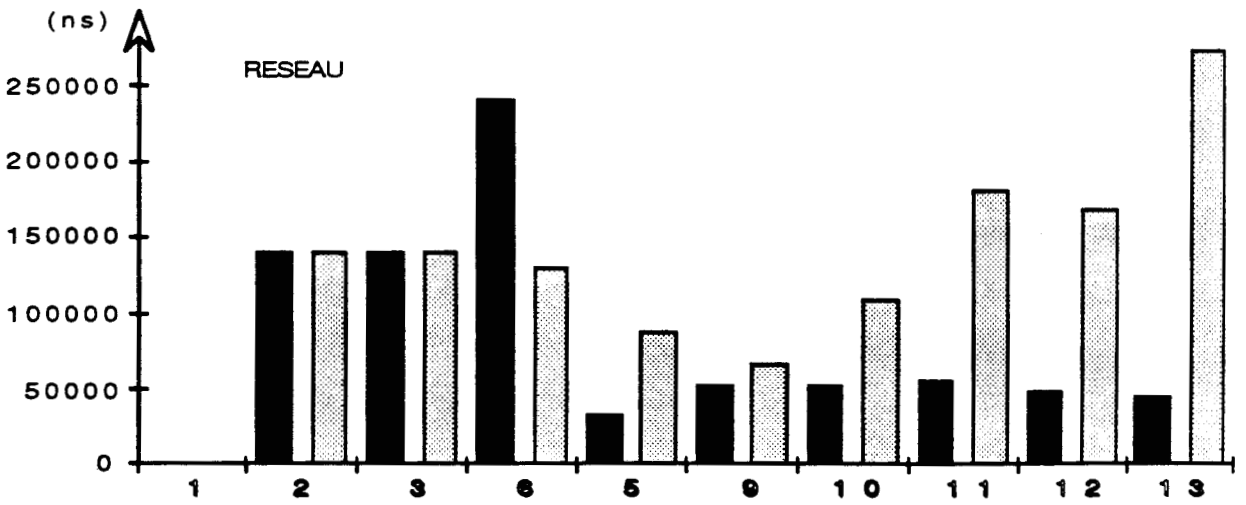
13

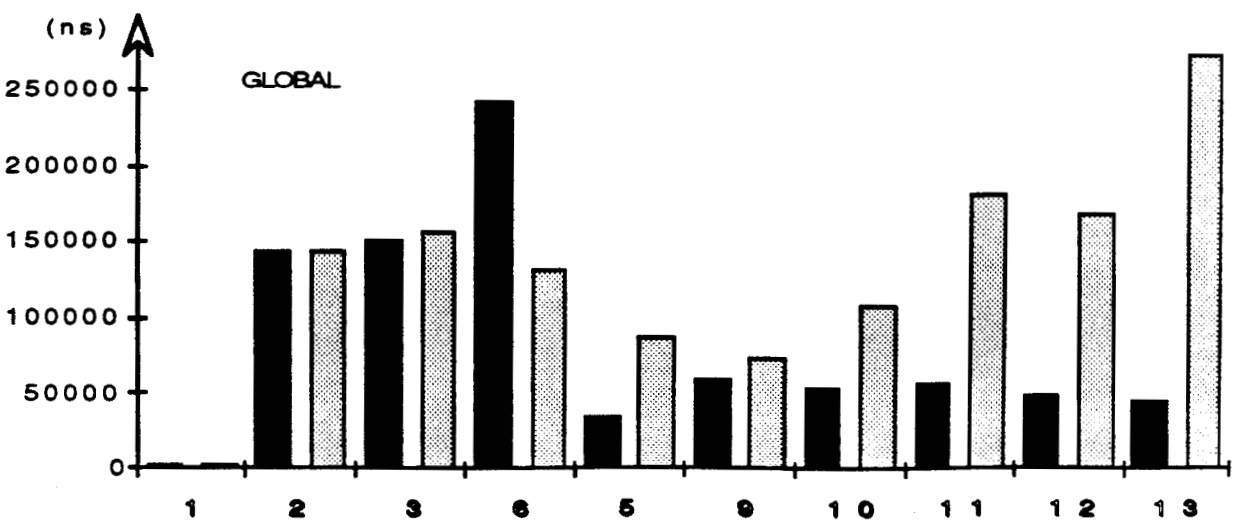
2*R = 20 => T_{cercle} = 46,1 µs
2*R = 300 => T_{cercle} = 273,4 µs

Comme pour le segment, l'évaluation telle quelle des algorithmes conduit inévitablement à choisir la solution SIMD, car le taux de parallélisme ne transparait pas au travers de ces chiffres.

TEMPS POUR 1 CERCLE







Nous allons maintenant tenter de comparer le parallélisme inhérent à chacune des solutions.

Pour cela, il nous faut faire un certain nombre d'hypothèses sur la composition de l'image. Nous avons choisi de comparer les temps nécessaires à la visualisation d'une scène composée de Nc cercles de rayon moyen R.

SIMD :

$$T = Ch + Th_A + (Nc-1) * SUP(Ch,Th,Cc) + Cc$$

MULTIPIPELINE :

$$Ne_m = \frac{ENT(\frac{N}{2R}) + ENT(\frac{N}{2*2R-1})}{2} = \frac{3}{4} * \frac{N}{2R}$$

$$T = Ne_m * Ch + Th + N * CTc + SUP((\frac{Nc}{Ne_m} - 1),0) * SUP(Ne_m * Ch,Th,CTc,Cc) + Cc$$

PROPAGATION PURE :

$$T = Ch + Th + N * CTc + (Nc-1) * SUP(Ch,Th,CTc,Cc) + Cc \quad \text{(4 voisins)}$$

ou

$$T = Ch + Th + N/2 * CTc + (Nc-1) * SUP(Ch,Th,CTc,Cc) + Cc \quad \text{(8 voisins)}$$

PROPAGATION LIMITEE :

$$Ne_m = \frac{ENT(\frac{N^2}{4R^2}) + ENT(\frac{N^2}{2*4R^2-1})}{2} = \frac{5}{8}\frac{N^2}{4R^2}$$

4 voisins :

$$T = Ne_m * Ch + SUP((\frac{Nc}{Ne_m} - 1),0) * (Ne_m * Ch,Th,Tp_r + 2R * CTc))$$

8 voisins :

$$T = Ne_m * Ch + SUP((\frac{Nc}{Ne_m} - 1),0) * (Ne_m * Ch,Th,Tp_r + R * CTc))$$

(+ test d'activité)

SUIVI DE CONTOUR :

$$N_{étapes} = ENT(\frac{2*\Pi*R}{NG}) + 1$$

$$T = Ne_m*Ch + SUP((\frac{Nc}{Ne_m} - 1),0) * (Ne_m*Ch,Th,Tp_r + Nétapes*CTc + Tp_S))$$

$$Ne_m = \frac{N^2}{100*NG}$$

Cc = Calcul dans la cellule
Ch = Calcul dans le hôte
CTc = Calcul dans la cellule avant la transmission + transmission
Th = Transmission du hôte vers le réseau
Th_A = Amorçage du pipeline de distribution (SIMD)
Tp_r = Protocole pour atteindre une cellule
Tp_S = Protocole de sortie du réseau (contrôle)

Ne_m = Nombre d'envois simultanés moyen

Rappel :
Pour l'évaluation du Ne_m du suivi de contour, nous autorisons la possibilité de conflits d'accès entre différents tracés. La valeur que nous donnons est donc le nombre de tracés qui peuvent être en cours simultanément sans que le réseau soit saturé.

Comme pour le segment, une machine hôte séquentielle ne peut pas alimenter le réseau correctement.

Pour l'évaluation qui suit nous allons, en conservant les hypothèses architecturales sur le réseau, exprimer quelles doivent être les performances du hôte pour obtenir une solution cohérente.

Nous évaluons les débits que l'on peut obtenir sur les différents réseaux, en faisant abstraction des performances du hôte.

SIMD

1 Tc=0 & Cc=640 ns ==> Débit = 1 cerc/640 ns = 1,56 M cerc/secondes

MULTIPIPELINE

2 CTc=140 ns & Cc=540 ns ==> Débit = Ne_m cerc/540 ns

2R = 20 => Ne_m = 32 Débit = 32 cerc/540 ns = 57,6 M cerc/secondes
2R = 300 => Ne_m = 2 Débit = 2 cerc/540 ns = 3,6 M cerc/secondes

3 CTc=160 ns & Cc=220 ns ==> Débit = Ne_m cerc/220 ns

2R = 20 => Ne_m = 32 Débit = 32 cerc/220 ns = 145 M cerc/secondes
2R = 300 => Ne_m = 2 Débit = 2 cerc/220 ns = 9 M cerc/secondes

PROPAGATION PURE

6

4voisins :
CTc=240 ns & Cc=340 ns ==> Débit = 1 cerc/340 ns = 2,9 M cerc/secondes

8 voisins :
CTc=280 ns & Cc=380 ns ==> Débit = 1 cerc/380 ns = 2,6 M cerc/secondes

PROPAGATION LIMITEE

5

4 voisins :
30000 + 320 * 2R ns
2R = 20 => Ne_m = 1500 Débit = 1500 cerc/36 µs = 4,1 M cerc/secondes
2R = 300 => Ne_m = 6 Débit = 6 cerc/126 µs = 4700 cerc/secondes

8 voisins :
30000 + 380 * R ns
2R = 20 => Ne_m = 1500 Débit = 1500 cerc/33 µs = 45 M cerc/secondes
2R = 300 => Ne_m = 6 Débit = 6 cerc/140 µs = 4300 cerc/secondes

SUIVI DE CONTOUR

$$N_{étapes} = ENT(\frac{2*\Pi*R}{NG}) + 1$$

9 Temps réseau = 40000 + 320 * $N_{étapes}$ + 10000 ns, NG=8, Ne_m = 1225

2R = 20 $N_{étapes}$ = 8 => Débit = 1225 cerc/52 µs = 24 M cerc/secondes
2R = 300 $N_{étapes}$ = 118 => Débit = 1225 cerc/87 µs = 14 M cerc/secondes

10 Temps réseau = 40000 + 380 * $N_{étapes}$ + 10000 ns, NG=8, Ne_m = 1225

2R = 20 $N_{étapes}$ = 8 => Débit = 1225 cerc/53 µs = 23 M cerc/secondes
2R = 300 $N_{étapes}$ = 118 => Débit = 1225 cerc/94 µs = 13 M cerc/secondes

11 Temps réseau = 40000 + 460 * $N_{étapes}$ + 10000 ns, NG=4, Ne_m = 2500

2R = 20 $N_{étapes}$ = 16 => Débit = 2500 cerc/57 µs = 44 M cerc/secondes
2R = 300 $N_{étapes}$ = 236 => Débit = 2500 cerc/158 µs
= 16 M cerc/secondes

12 Temps réseau = 30000 + 460 * $N_{étapes}$ + 10000 ns, NG=4, Ne_m = 2500

2R = 20 $N_{étapes}$ = 16 => Débit = 2500 cerc/47 µs = 53 M cerc/secondes
2R = 300 $N_{étapes}$ = 236 => Débit = 2500 cerc/148 µs
= 17 M cerc/secondes

1 3 Temps réseau = 30000 + 400 * $N_{étapes}$ + 10000 ns, NG=2, Ne_m = 5000

2R = 20 $N_{étapes}$ = 32 => Débit = 5000 cerc/53 µs = 99 M cerc/secondes
2R = 300 $N_{étapes}$ = 472 => Débit = 5000 cerc/229 µs
= 22 M cerc/secondes

Il faut maintenant tenter d'évaluer la faisabilité des machines hôtes associées, en fonction du nombre de calculs qu'elles ont à effectuer pour offrir un débit suffisant.

SIMD

1 640 ns pour 10 calculs
15,6 Mips => Débit = 1,56 M cerc/secondes

MULTIPIPELINE

2 540 ns pour 21*Ne_m calculs

2R = 20 => Ne_m = 32 640 calculs
1180 Mips => Débit = 57,6 M cerc/secondes

2R = 300 => Ne_m = 2 42 calculs
77 Mips => Débit = 3,6 M cerc/secondes

3 220 ns pour (23+2R*4)*Ne_m calculs

2R = 20 => Ne_m = 32 3300 calculs
15000 Mips => Débit = 145 M cerc/secondes

2R = 300 => Ne_m = 2 2400 calculs
10000 Mips => Débit = 9 M cerc/secondes

PROPAGATION PURE

6

4 voisins :
340 ns pour 10 calculs
29 Mips => Débit = 2,9 M cerc/secondes

8 voisins :
380 ns pour 10 calculs
26 Mips => Débit = 2,6 cerc/secondes

PROPAGATION LIMITEE

5

$10*Ne_m$ calculs

4 voisins :

2R = 20 => 36 µs Ne_m = 1500 15000 calculs
468 Mips => Débit = 4,1 M cerc/secondes

2R = 300 => 126 µs Ne_m = 6 60 calculs
0,4 Mips => Débit = 4700 cerc/secondes

8 voisins :

2R = 20 => 33 µs Ne_m = 1500 15000 calculs
454 Mips => Débit = 4,5 M cerc/secondes

2R = 300 => 140 µs Ne_m = 6 60 calculs
0,4 Mips => Débit = 4300 cerc/secondes

SUIVI DE CONTOUR

9 Ne_m = 1225

2R = 20 => 52 µs pour 58000 calculs
1100 Mips => Débit = 24 M cerc/secondes

2R = 300 => 87 µs pour 58000 calculs
660 Mips => Débit = 14 M cerc/secondes

10 Ne_m = 1225

2R = 20 => 53 µs pour 11000 calculs
210 Mips => Débit = 23 M cerc/secondes

2R = 300 => 94 µs pour 11000 calculs
120 Mips => Débit = 13 M cerc/secondes

11 Ne_m = 2500

2R = 20 => 57 µs pour 12500 calculs
220 Mips => Débit = 44 M cerc/secondes

2R = 300 => 158 µs pour 12500 calculs
79 Mips => Débit = 16 M cerc/secondes

12 Ne_m = 2500

2R = 20 => 47 µs pour 12500 calculs
261 Mips => Débit = 53 M cerc/secondes

2R = 300 => 148 µs pour 12500 calculs
84 Mips => Débit = 17 M cerc/secondes

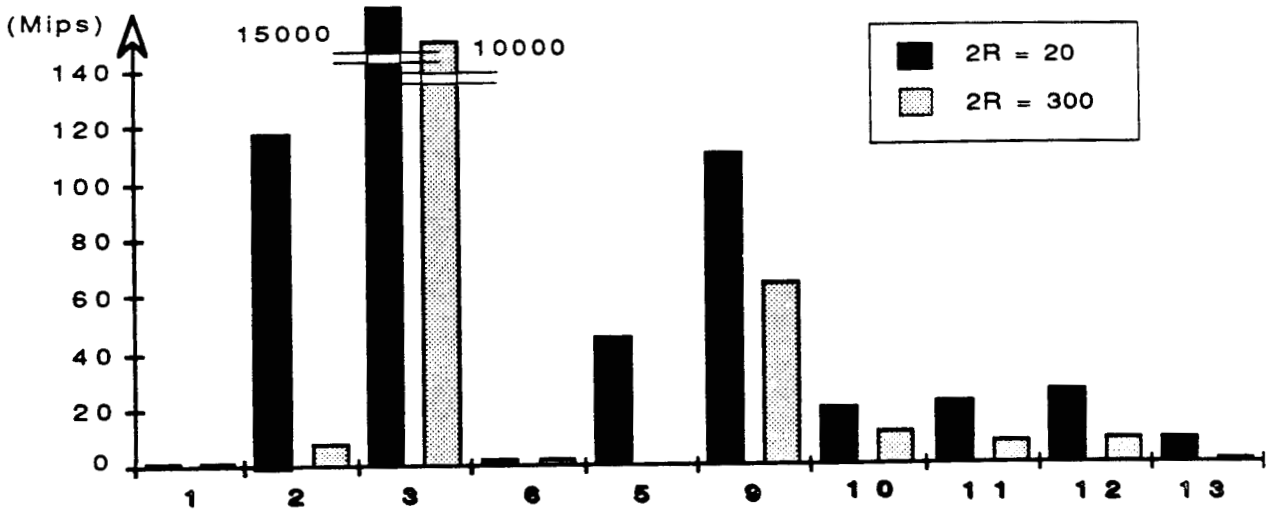
13 Ne_m = 5000

2R = 20 => 53 µs pour 5000 calculs
95 Mips => Débit = 99 M cerc/secondes

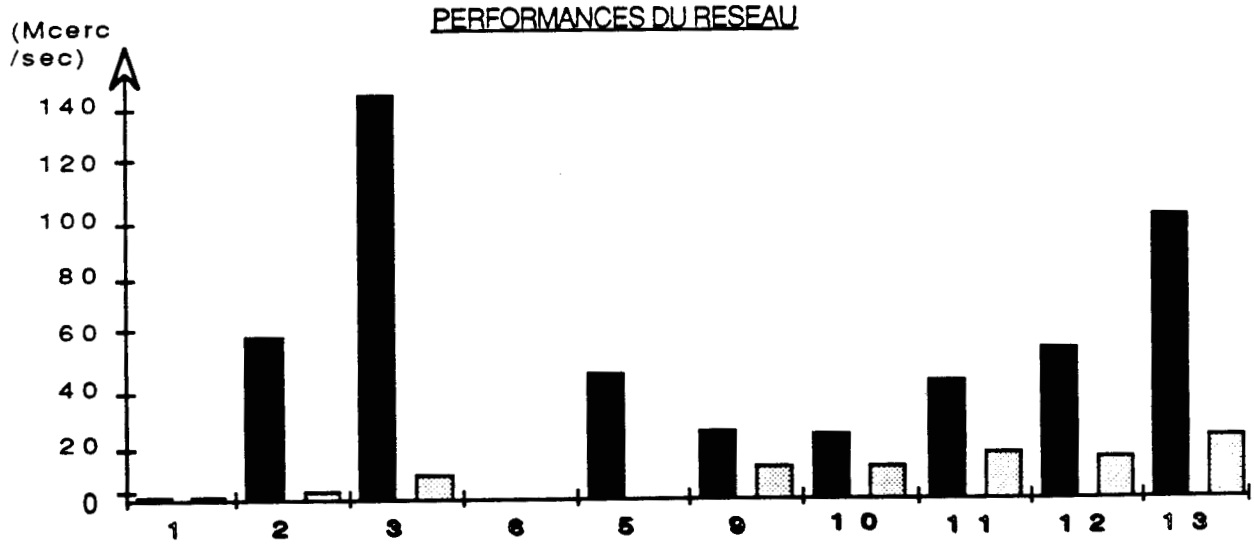
2R = 300 => 229 µs pour 5000 calculs
22 Mips => Débit = 22 M cerc/secondes

Algorithme N° :	Performances du hôte souhaitable (en Mips)	Performances du réseau attendues (en Mcerc/secondes)	Nombre d'instructions par cercle
1	15,6	1,56	10
2	2R = 20 => 1180 2R = 300 => 77	57,6 3,6	21 21
3	2R = 20 => 15000 2R = 300 => 10000	145 9	103 1111
6	4 voisins => 29 8 voisins => 26	2,9 2,6	10 10
5	2R = 20 => 454 2R = 300 => 0,4	45 0,004	10 10
9	2R = 20 => 1100 2R = 300 => 900	24 14	46 46
10	2R = 20 => 210 2R = 300 => 120	23 13	9 9
11	2R = 20 => 220 2R = 300 => 79	44 16	5 5
12	2R = 20 => 261 2R = 300 => 84	53 17	5 5
13	2R = 20 => 95 2R = 300 => 22	99 22	1 1

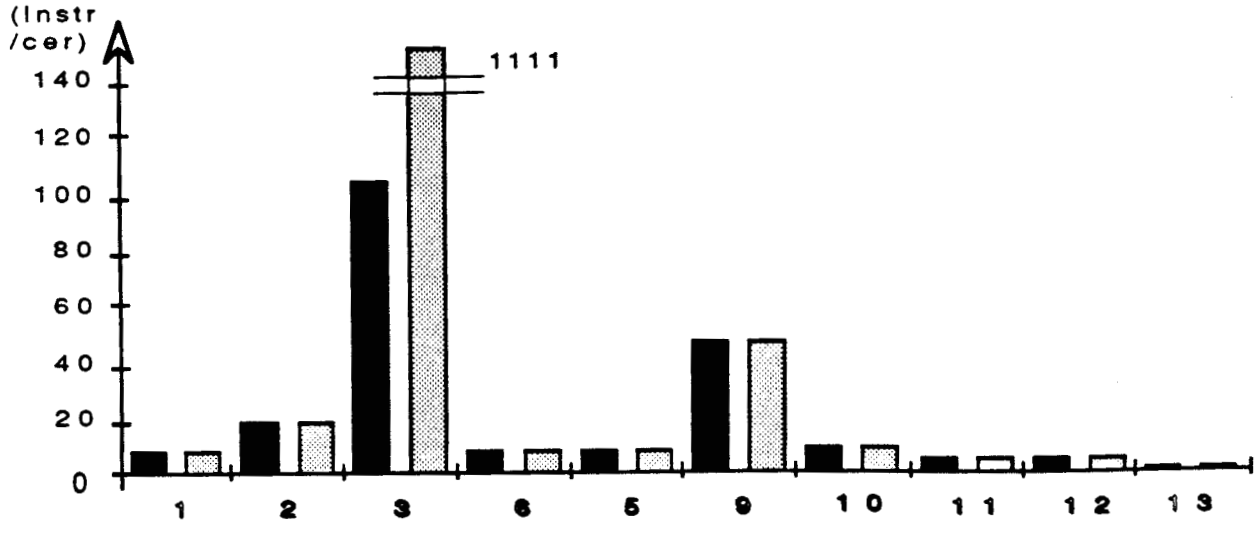
PERFORMANCES DU HOTE



PERFORMANCES DU RESEAU



NOMBRE D'INSTRUCTIONS PAR CERCLE



Remarques :

Le fait de ne pas s'intéresser aux conflits pour les solutions à SUIVI DE CONTOUR masque l'intérêt que l'on peut trouver à orienter la propagation.

D'autre part, le temps très important nécessaire au protocole de routage occulte le gain que l'on pouvait attendre d'une subdivision du cercle en octants. Il est préférable de tracer 8 cercles, en parallèle, en un temps plus long, que 8 cercles successifs divisés en octants.

Sous réserve d'une bonne gestion des conflits, les solutions à SUIVI DE CONTOUR, avec peu de germes, semblent les plus indiquées.

La solution MULTIPIPELINE simple 2 est elle aussi très intéressante, par contre l'utilisation du vecteur d'entrées 3 est peu recommandée.

Les approches SIMD 1 et par PROPAGATION PURE 6 sont moins performantes, mais plus facilement réalisables.

Enfin, la PROPAGATION LIMITEE 5 est peu gourmande en temps hôte, mais décevante en ce qui concerne les résultats.

Rappels :

Comme pour le segment c'est le nombre d'instructions par cercles qu'il faut examiner plutôt que les performances du hôte souhaitée. Certaines solutions se prêtent plus ou moins bien à la parallélisation :

- ◊ La solution SIMD 1 est relativement facile à satisfaire car les exigences du réseau ne sont pas excessives. Il suffit de multiplier les machines de précalculs (dans des proportions assez faibles) et de leur donner accès successivement à l'architecture de distribution.

- ◊ Pour les MULTIPIPELINE 2 3 il y a deux problèmes : l'un posé par le fait que le nombre de calculs est fonction de la longueur des segments (conduisant à des Nem différents), l'autre est un problème d'accès physique au réseau (pour l'envoi de plusieurs commandes simultanées).
  L'incrémentation du paramètre dans le vecteur d'entrées (plus complexe que pour le segment) peut être faite en parallèle par rapport au reste des calculs, réduisant d'autant la charge du hôte.

- ◊ Pour la PROPAGATION PURE 6 la solution est, comme en SIMD, dans la multiplication des machines hôte, mais dans un facteur plus élevé.

- ◊ La PROPAGATION LIMITEE 5 et le SUIVI DE CONTOUR 9 10 11 12 13 conduisent à un petit nombre de calculs mais pour de nombreux cercles. La parallélisation est simple étant donné l'indépendance des calculs, mais le taux de parallélisme obtenu est directement lié à la complexité architecturale, et sera par conséquent limité.

Nous étudierons plus en détail ces aspects dans la dernière partie de notre étude.

BIBLIOGRAPHIE:

[AKN 85], [BIS 85], [BRE 77], [CAR 77], [DAN 70], [FIE 83], [FIE 83], [FPW 87], [FRE 80], [HAN 86], [HEG 85], [HJL 73], [ILR 83], [KKS 79], [MER 84], [MOR 76], [NWS 81], [PEL 85], [PER 88], [PIE 86].

4. LE REMPLISSAGE

TABLE DES MATIERES

4. LE REMPLISSAGE

4.1. GENERALITES

Il y a quelques années encore l'utilisateur de C.A.O., D.A.O. se satisfaisait d'une image filaire comme représentation de son espace de travail.

Désormais, dans la plupart des cas, il exige une visualisation plus réaliste. Dans ce contexte, on a souvent recours au remplissage simple dans un premier temps, car les algorithmes de rendu très réaliste sont coûteux en temps de calcul. Les performances de ces algorithmes doivent être particulièrement bonnes, eu égard à leur fréquence d'utilisation, notamment lors des étapes intermédiaires de l'élaboration d'une image synthétique.

A priori le problème du remplissage de taches semble trivial, mais il n'en est rien; pour preuve le nombre de publications parues à ce sujet [HEG 85], [KIL 87], [LUC 82], [PAV 82], [PER 88], [TAN 88]...

Dans le souci de conserver des performances acceptables, certains algorithmes ne sont applicables qu'à une classe bien définie de contours (par exemple : les polygones convexes). D'autre part, il n'est pas rare de rencontrer des algorithmes qui fournissent des résultats erronés pour quelques figures particulières.

Enfin, il faut distinguer deux classes de problèmes:

- Le remplissage de contours préinscrits en mémoire (sous forme d'un ensemble supposé connexe de pixels).
- Le remplissage de contours à partir de leur donnée formelle (suite de sommets, équations...).

4.2. LE PREDECOUPAGE

Une manière de pallier la complexité des algorithmes de remplissage, est d'effectuer un prédécoupage de la tache en surfaces élémentaires faciles à remplir Le remplissage de ces dernières peut alors être confié à un processeur spécialisé.

Par exemple, certains algorithmes très performants sont spécifiques aux polygones convexes. Les polygones non convexes sont subdivisés en polygones convexes afin de pouvoir être remplis.

RAPPEL : Un polygone convexe est tel que tous les segments joignant deux à deux les sommets du polygone sont totalement internes.

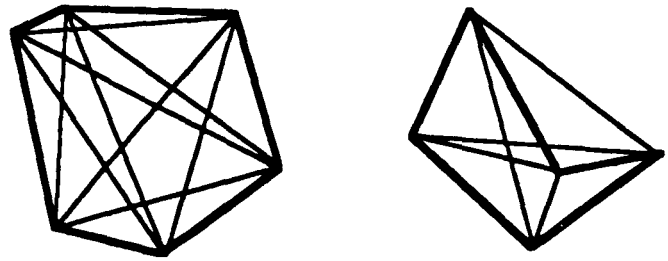


Figure 1 : Polygone convexe *Polygone concave*

Les principaux types de prédécoupages utilisés sont [OUL 88] :

- Découpage en triangles [FOM 84], [HER 83], [WOO 85]
- Découpage en trapèzes [BRF 79], [CHA 84]
- Découpage en polygones convexes [SCH 78] [LIT 79]

Le problème du découpage ne sera pas développé plus en détail dans ces pages, car il s'agit par essence d'un précalcul, qui, de plus, est centralisé. Or l'intérêt des machines parallèles est justement de permettre la distribution des tâches aux différents processeurs.

Le prédécoupage conduit à une surcharge de l'ordinateur hôte d'où des performances moindres, même si par ailleurs les surfaces élémentaires sont remplies en parallèle.

4.3. LES ALGORITHMES SEQUENTIELS

Les algorithmes les plus courants forment trois classes distinctes :

- Algorithmes basés sur le nombre d'intersections des lignes d'écran avec le contour (PARITY CHECK).
- Algorithmes issus de l'étude de la position des points par rapport aux différentes arêtes (EDGE FILLING).
- Algorithmes de remplissage par propagation à partir d'un ou plusieurs points intérieurs au contour (SEED FILLING).

D'autres algorithmes utilisent un mélange de plusieurs de ces méthodes : par exemple, positionnement et parité [ACK 81]. Ces derniers n'apportent pas de progrès vraiment significatif.

Remarque :
Comme c'est souvent le cas dans la bibliographie, nous présentons des algorithmes pour le remplissage de polygones quelconques. Nous signalerons pour chacun d'entre eux dans quelle mesure il est possible d'étendre leur application à d'autres figures.

4.3.1. TEST DE PARITE

[LUC 82] [PAV 82]

Tout d'abord définissons ce qu'est un point intérieur au contour : Un point est considéré comme intérieur à une zone si toute demi-droite issue de ce point possède un nombre impair d'intersections avec le contour.

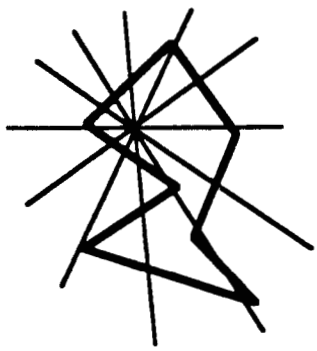


Figure 2 : Point intérieur

L'algorithme découle tout naturellement de cette définition : En suivant les lignes de balayage écran, le décompte des intersections, avec la figure, déjà rencontrées indique si le point atteint est intérieur ou extérieur au contour.

{ cf ANNEXE C : REMPLISSAGE SEQUENTIEL 1 }

Mais, un algorithme directement dérivé de ce principe possède beaucoup d'imperfections. La principale d'entre elles est que de nombreuses figures induisent des résultats erronés.

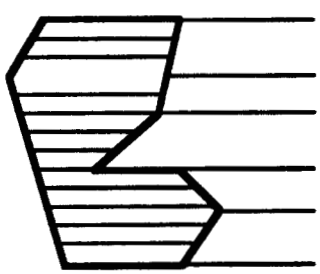
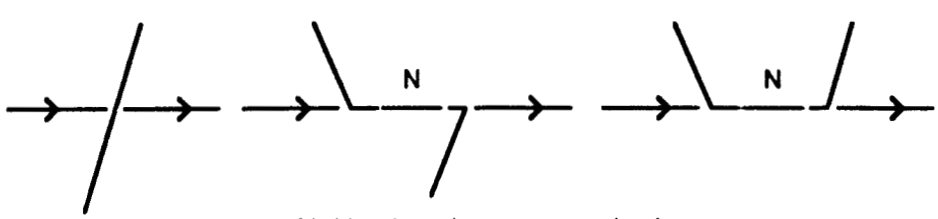


Figure 3 : Remplissage incorrect

En pratique, les différentes situations rencontrées peuvent se résumer en [ISO 73] :



N: Nombre de segments horizontaux

Figure 4 : Les différents cas

Le premier cas qui semble ne pas poser de problème, est à examiner à un niveau plus fin, celui du pixel. L'accent doit être mis sur les ambiguïtés soulevées par les pixels situés sur une même ligne de balayage.

Figure 5 : Pixels voisins alignés

De plus, il se peut que deux lignes soient confondues en un même point, sans pour cela qu'il s'agisse de polygones croisés qui ne sont pas pris en compte dans notre étude. C'est le cas par exemple, lorsque deux vecteurs sont très proches et se superposent sur une même suite de points de l'image.

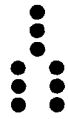


Figure 6 : Segments en partie confondus

Les différentes façons de résoudre ces problèmes sont regroupées en trois classes :

4.3.1.1. INSCRIPTION DIRECTE

[LUC 82] [NWS 79]

Les algorithmes de remplissage à partir de la description formelle du contour sont le plus souvent dérivés de l'algorithme YX exposé dans [NWS 79].

Cette méthode comprend trois étapes :

- Pour chacun des cotés du polygone déterminer les intersections avec les lignes de balayage de l'écran. Cela est obtenu grâce à un algorithme de génération de vecteurs, par exemple : algorithme de BRESENHAM.

- Trier ensuite, dans l'ordre croissant, tous les couples de coordonnées ainsi obtenus, en respectant la relation d'ordre :

$$(X_1,Y_1) > (X_2,Y_2) \iff (Y_1>Y_2) \text{ ou } ((Y_1=Y_2) \& (X_1>X_2))$$

- Enfin, parcourir cette liste en prenant des paires de points consécutifs, et tracer les segments les joignant. Par construction, tous les points des segments ainsi définis sont intérieurs au contour et sont donc affichés.

Voici comment sont résolus les problèmes posés par les singularités du tracé :

Les problèmes liés au tracé du segment proprement dit, tel que la création de plusieurs points de même ordonnée, se résolvent simplement en modifiant la génération de vecteur afin de ne conserver qu'un seul de ces points (le plus extérieur à la tache). Il y a deux choix possibles : soit on considère le remplissage de l'intérieur du contour, soit le remplissage d'une tache (intérieur + contour). C'est cette dernière option que nous retenons pour notre présentation.

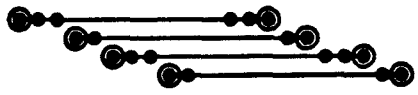


Figure 7 : Un seul point conservé par ligne

Le point le plus épineux concerne les cas :

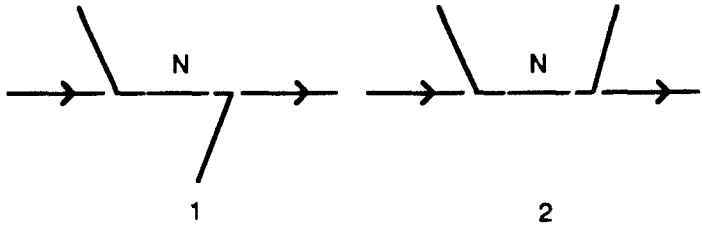


Figure 8 : Segments horizontaux

Ces deux figures illustrent en fait plusieurs problèmes:

i) Les jonctions de segments (cas 1 avec N=0) :

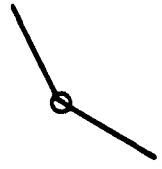


Figure 9 : Point d'inflexion

ii) Les pointes (cas 2 avec N=0) :

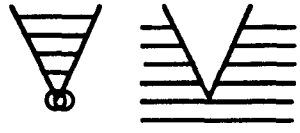


Figure 10 : Pointes

iii) Les segments horizontaux :

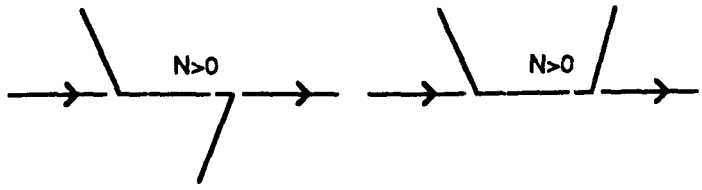


Figure 11 : Segments horizontaux

Quatre cas se présentent :

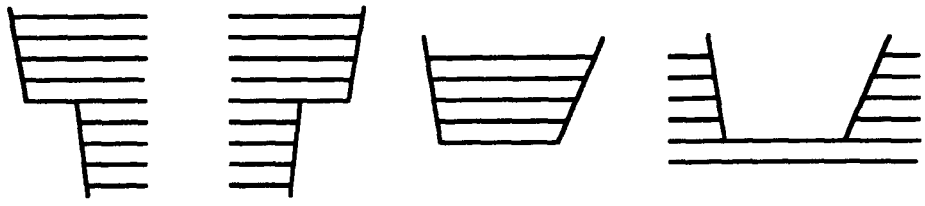


Figure 12 : Cas possibles

Pour résoudre ces problèmes, il faut se fixer un ordre lors de l'élaboration de la liste, par exemple l'ordre trigonométrique, afin de déterminer de quel cas de figure il s'agit.

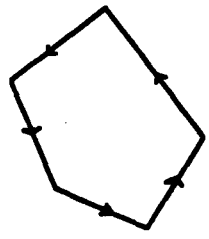


Figure 13 : Orientation du contour

Pour résoudre notre problème, nous ne conservons pas les segments horizontaux :

- Il faut, suivant le cas, conserver soit le dernier point du segment précédent, soit le premier point du suivant :

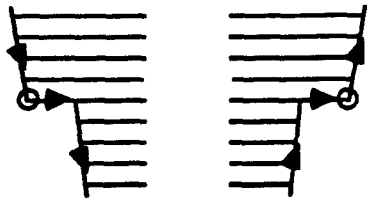


Figure 14 : Inflexion du contour

- Il faut à la fois conserver le dernier point du segment précédent et le premier point du suivant :

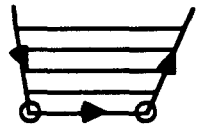


Figure 15 : Pointe pleine

- Dans le quatrième cas de figure on peut ne conserver aucun point, puisque le contour est recouvert par un segment de remplissage :

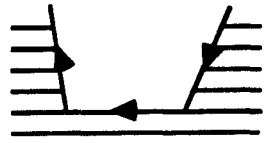


Figure 16 : Pointe creuse

{ cf ANNEXE C : REMPLISSAGE SEQUENTIEL 2 }

Remarques:
- Cet algorithme nécessite un précalcul important et centralisé.
- Il requiert un espace mémoire très important pour conserver la liste de points.

Extension à des figures plus complexes : sans problème, sous réserve de la réécriture des primitives de tracé capables de trier les points pour ne conserver que ceux utiles.

4.3.1.2. INSCRIPTION APRES MARQUAGE

[LUC 82]

Le marquage s'effectue selon la règle suivante :

- Ecrire la valeur 1 si le passage par ce point implique un changement de parité.
- Ecrire 2 si le passage n'implique pas de changement.

Remarque :
De même que pour l'inscription directe, il faut utiliser un algorithme dérivé d'un tracé de vecteurs pour pouvoir:

- Traiter les points de même ordonnées,
- Traiter les pointes,
- Traiter les intersections...

En fait, le marquage est assez proche du stockage dans la liste de l'algorithme précédent. Au lieu de conserver les points intéressants dans une liste, la marque 1 leur est affectée, les autres points ont la marque 2.

Notons quand même que les points confondus requièrent un traitement particulier :

Si le point atteint a déjà été marqué, il faut :
- mettre sa marque à 1 si l'une des deux marques, la nouvelle ou l'ancienne, est à 1,
- la mettre à 2 si les deux marques sont identiques.

Ceci, s'il ne s'agit pas d'une extrémité de segment traitée par ailleurs.

Une fois le marquage effectué le procédé de remplissage est trivial.

{ cf ANNEXE C : REMPLISSAGE SEQUENTIEL 3 }

Les avantages de cette solution par rapport à la précédente sont :

- La réduction du précalcul,
- Le remplissage qui peut être différé par rapport au tracé du contour,
- L'utilisation de la mémoire d'image pour stocker les informations plutôt qu'une autre structure (liste).

Les inconvénients sont d'ailleurs liés à ce dernier point :

- Elle nécessite deux passes et donc deux séries d'accès à la mémoire de trame,
- L'utilisation de la mémoire de trame pose des problèmes si celle-ci est non vide.

Extension à des figures plus complexes : Sans problème à condition de réécrire les primitives de tracé.

4.3.1.3. CONTOUR PREINSCRIT

[PAV 82]

Nous allons présenter deux méthodes :

i) PREMIER CAS :

La résolution des particularités se fait par l'observation des points du contour du voisinage dont l'appartenance ou la non appartenance a déjà été déterminée.

{ cf ANNEXE C : REMPLISSAGE SEQUENTIEL 4 }

Remarque :
L'examen du voisinage induit un parcours de la mémoire de trame qui grève sensiblement le temps d'exécution. C'est essentiellement pour cela qu'est développée la deuxième solution qui n'engendre pas ce type de phénomène.

Extension à des figures plus complexes : Le procédé s'applique sans modification au remplissage de tout contour fermé.

ii) DEUXIEME CAS :

C'est un dérivé lointain de la méthode YX exposé dans [NWS 79]. Une solution de ce type est proposée dans [GOP 88].

En premier lieu il faut se définir une numérotation pour les différentes directions :

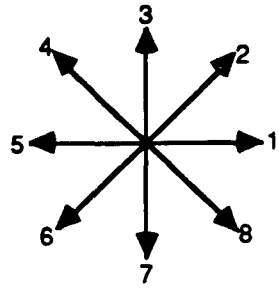


Figure 17 : Numérotation des directions

Ensuite se définir un sens de parcours, par exemple le sens trigonométrique.

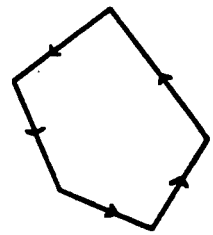


Figure 18 : Orientation du contour

Déterminer le point supérieur gauche du contour (élémentaire grâce à un balayage haut-bas, gauche-droite de l'image).

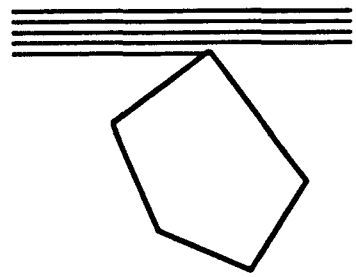


Figure 19 : Recherche du point supérieur gauche

Le remplissage se fera par tracés successifs de segments horizontaux orientés de la gauche vers la droite :

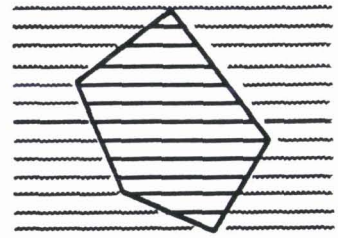


Figure 20 : Remplissage

Connaissant le sens de parcours, la direction du point précédent et la direction du point suivant, il est possible de définir une table logique permettant de savoir, pour chaque point, s'il est :

- origine d'un segment de remplissage,
- extrémité d'un segment de remplissage,
- point inutile au remplissage.

Etant donné les directions que nous avons choisies, la table est :

DS

DP \ DS	1	2	3	4	5	6	7	8
1	X	X	X	X	D	D	D	D
2	X	X	X	X	D	D	D	D
3	X	X	X	X	D	D	D	D
4	A	A	A	A	X	X	X	X
5	A	A	A	A	X	X	X	X
6	A	A	A	A	X	X	X	X
7	A	A	A	A	X	X	X	X
8	X	X	X	X	D	D	D	D

D : Point de départ de segment
A : Point d'arrivée de segment
X : Point inutile ou cas impossible

DS : Direction du successeur
DP : Direction du prédécesseur

Reste donc à déterminer pour chaque point du contour le couple (DS,DP). Ceci est fait en parcourant le contour dans le sens défini au départ. Pour déterminer le successeur du point, l'ordre d'examen de ses voisins dépend de la direction de son prédécesseur.

En fait, la scrutation du voisinage se fait dans le sens trigonométrique à partir du point prédécesseur.

Figure 21 : Scrutation du voisinage

Si aucune des sept cellules n'appartient au contour, c'est qu'il s'agit d'une pointe, donc que le point prédécesseur est également le successeur :

Figure 22 : Pointe

{ cf ANNEXE C : REMPLISSAGE SEQUENTIEL 5 }

Remarque :
Cette solution nécessite, par deux fois, le tri d'une liste de taille importante.

Extension à des figures plus complexes : Aucune modification à apporter.

4.3.2. REMPLISSAGE PAR COMPLEMENTATION (EDGE FILLING)

[ACK 81] [DUN 83] [LAN 83]

L'idée de l'algorithme est élémentaire : Pour chaque point de l'image déterminer sa position relative par rapport à chacune des arêtes du contour.

De par son principe, cet algorithme est principalement réservé au remplissage de contours connus à partir de leurs définitions formelles.

En pratique, pour chaque côté du polygone, on complémente les pixels situés sur la partie droite des lignes d'écrans coupant ledit segment.

Remplissage de :

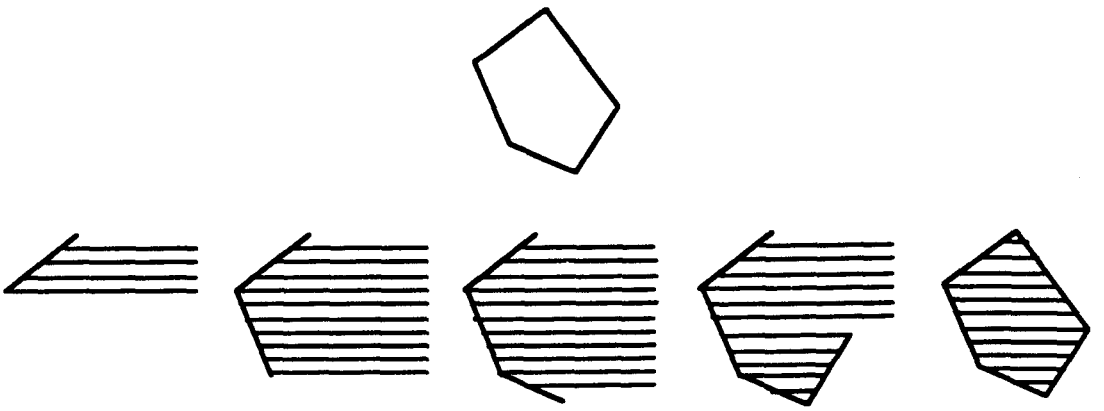


Figure 23 : Remplissage par complémentation

Il ne suffit pas, pour chaque point du segment affiché de complémenter les points situés sur sa droite. Le traitement des segments horizontaux nécessite un traitement particulier, il y a également des précautions à prendre pour les points de même ordonnée sous peine d'obtenir des remplissages incorrects.

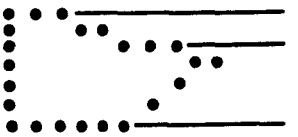


Figure 24 : Remplissage incorrect

Les algorithmes sont donc des variantes des algorithmes de génération de segments, ce qui induit une grande diversité lors de l'implantation effective.

En fait, les problèmes évoqués lors de l'étude du remplissage par test de parité se retrouvent ici. La manière de les résoudre est identique.

Rappel :

Pour remplir correctement un contour, il est nécessaire de ne traiter que les points les plus extérieurs à la tache, ce qui requiert l'orientation préalable du contour.

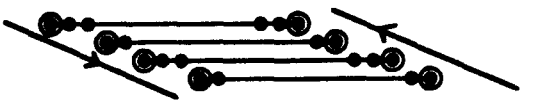


Figure 25 : Points extérieurs

{ cf ANNEXE C : REMPLISSAGE SEQUENTIEL 6 }

Remarque :

Cette méthode permet de remplir les polygones non convexes et même les polygones troués. Pour ces derniers, il suffit de les orienter en sens inverse par rapport aux polygones pleins.

Remplir :

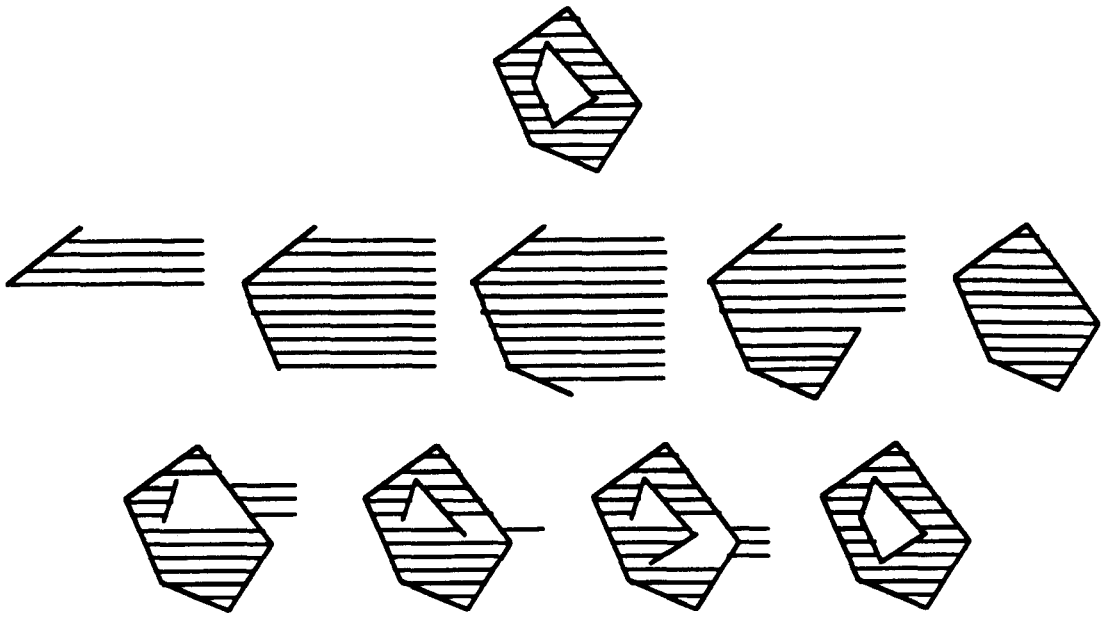


Figure 26 : Polygone troué

Remarques :

- L'ordre d'élaboration des différents segments n'a pas d'importance.
- L'utilisation de la complémentation suppose l'image initialement vierge, ou qu'une valeur puisse être réservée pour chaque pixel.

Extension à des figures plus complexes : Sans problème sous réserve de réécrire les procédures de tracé qui lanceront des commandes de complémentation à partir des points utiles.

4.3.3. REMPLISSAGE A PARTIR D'UN POINT GERME (SEED FEEL)

[LUC 82] [PEL 85]

C'est sans doute le plus intuitif : Un point est intérieur à la tache si aucun de ses voisins directs n'est extérieur.

L'algorithme est basé sur le corollaire de cette assertion : Etant donné un point intérieur, ses voisins ne sont pas extérieurs, tout au plus appartiennent-ils au contour.

Tout point possédant la propriété d'être intérieur transmet ladite propriété à ses voisins. La propagation s'arrête quand il s'agit d'un point du contour.

Reste à résoudre le problème de la propagation. Pour ce faire, nous présentons deux méthodes :

4.3.3.1. PROPAGATION LINEAIRE

[SMI 79]

La propriété se propage ligne à ligne. Une pile mémorise les directions qui n'ont pas encore été explorées.

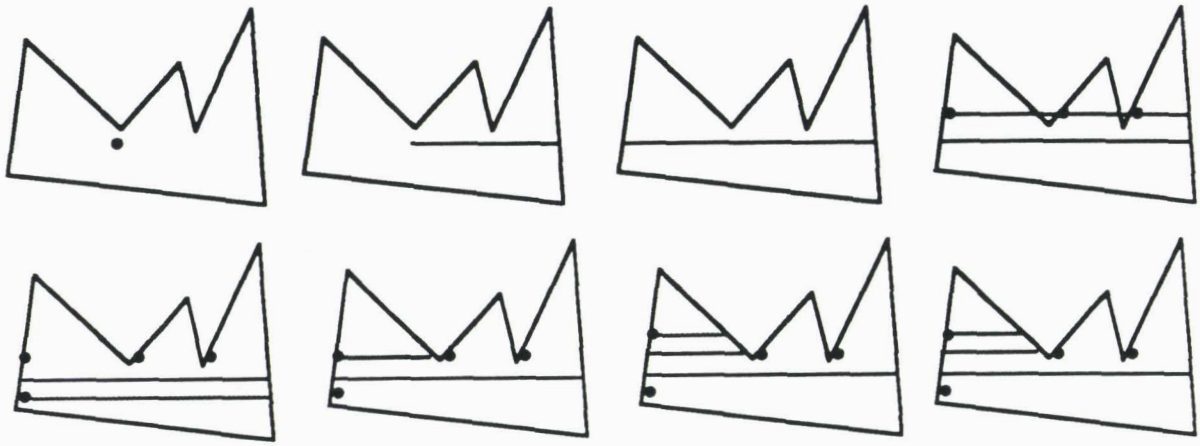


Figure 27 : Propagation linéaire

{ cf ANNEXE C : REMPLISSAGE SEQUENTIEL 7 }

Remarques :

Certaines figures très particulières ne sont pas remplies correctement.

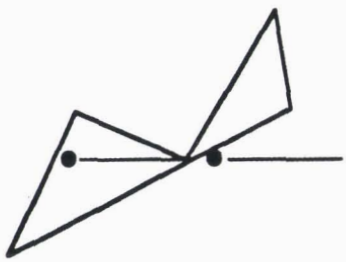


Figure 28 : Cas particulier

Ceci est dû à la recherche des points à empiler qui parcourt les points du contours colinéaires successifs sans les différencier.

Extension à des figures plus complexes : Sans aucune modification.

4.3.3.2. DIFFUSION A TOUS LES VOISINS

Cette solution consiste à propager la propriété d'être intérieur aux voisins directs, ceci par le truchement d'une pile.

Figure 29 : quatre voisins huit voisins

Chaque point dont un voisin est intérieur, est considéré comme intérieur s'il n'appartient pas lui-méme au contour.

{ cf ANNEXE C : REMPLISSAGE SEQUENTIEL 8 }

Remarques :

- Cette solution a pour inconvénient majeur de solliciter fortement la pile.
- Par contre, elle ne peut être prise en défaut par des tracés particuliers.

Extension à des figures plus complexes : Sans modification.

Notes concernant ces deux approches :

◊ Il s'agit d'algorithmes de remplissage de contours préinscrits. Ils remplissent correctement les contours fermés définis à l'écran, mais cela peut ne pas correspondre au polygone mémorisé :

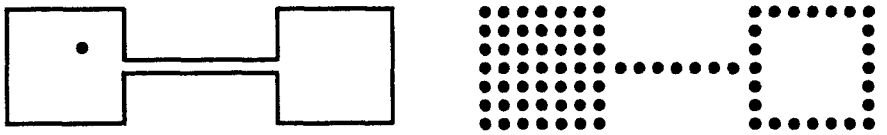


Figure 30 : Remplissage incomplet

◊ D'autre part l'algorithme suppose la donnée d'un point intérieur, ce qui le destine plus particulièrement aux logiciels graphiques interactifs. Le problème de la recherche d'un point intérieur par programme n'est en effet pas simple. Pour vérifier qu'un point est intérieur au contour, une solution consiste à utiliser le test de parité. Si le nombre d'intersections avec le contour des demi-droites issues de ce point est impair, le point est intérieur [PER 88]. Mais les cas particuliers à traiter rendent l'algorithme assez complexe (cf 4.3.1.1.).

4.4. LES ALGORITHMES PARALLELES

Certains algorithmes séduisants pour un usage séquentiel, sont totalement inefficaces une fois adaptés aux architectures cellulaires. Notamment tout ceux qui nécessitent un prétraitement centralisé important sont à proscrire; le problème a d'ailleurs déjà été évoqué au sujet du prédécoupage.

4.4.1. ALGORITHME BASE SUR L'EQUATION

[FUC 85]

Pour cette solution il suffit de modifier légèrement la génération des segments. Chaque cellule calcule sa position par rapport aux arêtes (grâce à l'équation), et décide si elle est située du même coté que le polygone à remplir. L'intérieur du polygone est constitué par l'ensemble des cellules situées du même coté que le polygone pour chacune de ses arêtes.

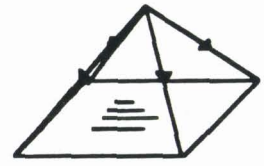


Figure 31 : Diffusion de l'équation

{ cf ANNEXE C : REMPLISSAGE PARALLELE 1 }

Remarque :

Cette solution ne convient que pour les polygones convexes. Une solution similaire peut être développée pour les polygones non convexes, toujours par la combinaison de demi-plans, mais elle nécessite un précalcul important. Ce dernier est une analyse détaillée des englobants des concavités rencontrées, permettant de définir la manière d'assembler les demi-plans.

4.4.2. ALGORITHMES BASES SUR LE TEST DE PARITE

Il faut discerner les trois approches possibles :

- remplissage direct,
- remplissage après marquage,
- remplissage d'un contour préinscrit.

4.4.2.1. REMPLISSAGE DIRECT

Les différentes étapes sont :

- Elaboration de la liste ordonnée des extrémités de segments de remplissage (réalisée par l'ordinateur hôte) comprenant :

  - Le calcul, sans affichage, des points du contour, en vue de sauvegarder ceux qui sont utiles au remplissage (par un algorithme de type BRESENHAM par exemple)
  - Le tri de cette liste

- Affichage des segments de remplissage (exécuté en parallèle sur le réseau). Remarquons qu'il s'agit de tracer des segments horizontaux; les algorithmes s'en trouvent considérablement simplifié.

  Pour ce faire, différents types de fonctionnement peuvent être envisagés :

  - Propagation contrôlée par les données, avec un protocole de routage.

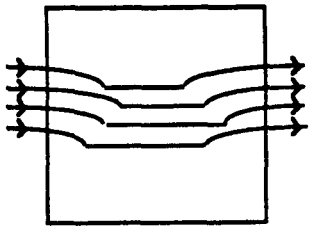


Figure 32 : Affichage par propagation

Etant donné le grand nombre de segments à tracer, cette solution semble peu intéressante. De plus, elle se rapproche alors sensiblement de la solution multipipeline.

- SIMD

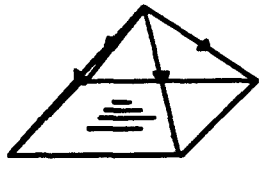


Figure 33 : Affichage SIMD

Le caractère séquentiel de l'émission des commandes est incompatible avec la multitude de segments à afficher.

- Multipipeline

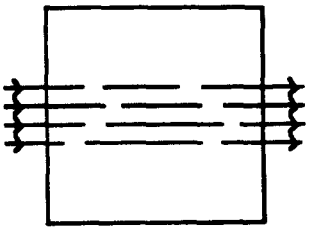


Figure 34 : Affichage multipipeline

Deux solutions s'offrent alors à nous:

i) Soit injecter une commande unique pour tracer plusieurs segments :

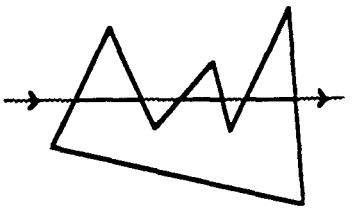


Figure 35 : Une seule commande

ii) Soit envoyer plusieurs commandes successives :

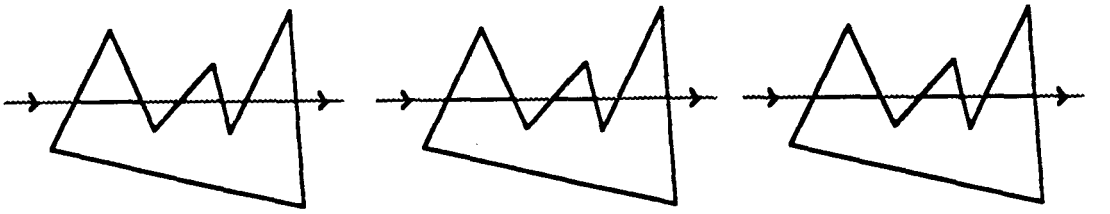


Figure 36 : Une commande par segment

La première solution complexifie par trop la commande transmise de cellule en cellule. Nous lui préférons la seconde.

La solution la plus appropriée est donc celle de l'algorithme multipipeline à commandes successives.

{ cf ANNEXE C : REMPLISSAGE PARALLELE 2 }

Remarque :

La construction de la liste, bien que relativement simple, nécessite un espace mémoire et un temps de calcul très importants. Comme celle-ci est confiée à l'ordinateur hôte, les performances de cet algorithme sont peu probantes, même si par ailleurs tous les segments sont affichés simultanément. Plus encore que pour les autres solutions, il nous sera impossible dans ce cas de réaliser une machine hôte capable d'alimenter correctement le réseau. C'est le temps passé par l'ordinateur hôte à construire puis trier la liste, qui déterminera les performances.

4.4.2.2. REMPLISSAGE APRES MARQUAGE

Le remplissage comprend deux phases :

- Inscription du contour avec un marquage,
- Afficher les segments entre les marques.

Remarques :

- Une partie du marquage ne peut être distribuée car elle nécessite la connaissance globale du contour. En particulier pour résoudre les cas :

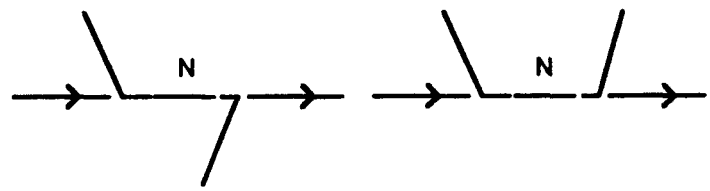


Figure 37 : Segments horizontaux

- Par contre, au niveau des vecteurs, les tracés avec marquage sont effectués en parallèle, ceci quelle que soit la méthode choisie (se référer au tracé de segments).

Il y a donc trois étapes :

◊ Repérage des singularités du contour : les pointes, les segments horizontaux (exécuté par l'ordinateur hôte). Nous avons remarqué lors de la présentation de l'algorithme sous forme séquentielle, que les extrémités des segments posent des problèmes.

Deux remèdes à cela :

- Afficher les segments sans leurs extrémités, puis envoyer des commandes d'affichage spéciales pour ces dernières.

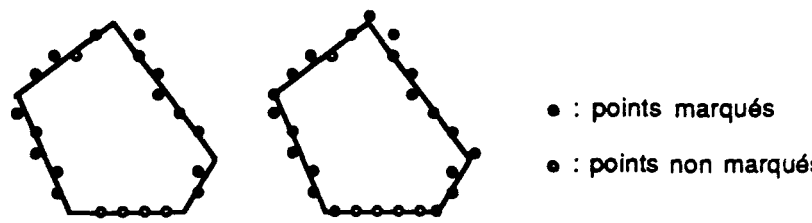


Figure 38 : Affichage séparé des extrémités

Cette première hypothèse permet d'envoyer toutes les commandes de tracé de contour en parallèle, mais nécessite l'émission de commandes supplémentaires pour les extrémités.

- Ajouter un paramètre à la commande indiquant quelles extrémités marquer.

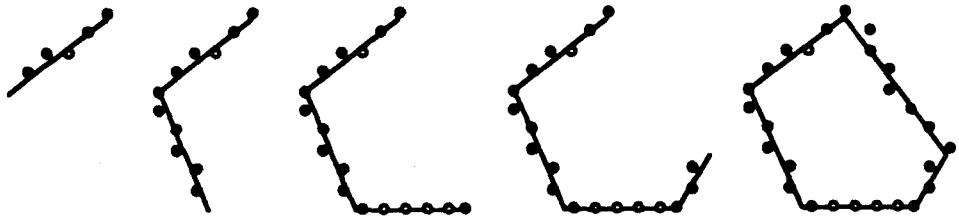


Figure 39 : Commandes différentes suivant le nombre d'extrémités utiles

L'évaluation des paramètres des commandes est nécessairement séquentielle, mais est peu gourmande en temps machine.

◊ Deuxième étape, tracer le contour avec les marques (par le réseau). Pour marquer le contour le fonctionnement peut être soit de type propagation, soit multipipeline. Mais il est préférable d'adopter le mode multipipeline pour afficher les segments de remplissage, comme nous l'avons vu pour le remplissage direct. La solution multipipeline semble donc toute indiquée pour effectuer le tracé avec marquage.

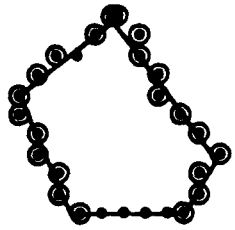


Figure 40 : Tracé du contour (multipipeline)

Dans le cas multipipeline, pour les problèmes des points d'un même segment situés sur une même ligne, nous ne conservons que :

- le premier point du segment rencontré, sur une ligne de balayage, pour un segment orienté de haut en bas,
- le dernier point du segment rencontré, sur une ligne de balayage, pour un segment orienté de bas en haut.

Le contour étant parcouru dans le sens trigonométrique.

◊ Enfin troisième étape, remplissage proprement dit (en parallèle sur le réseau). Le choix du mode multipipeline est dicté par les mêmes remarques que pour la solution précédente. Avec, en outre, le fait que pour effectuer le remplissage, le marquage doit être terminé, donc un contrôle d'exécution serait nécessaire, mais pour le multipipeline l'ordre d'émission fixe l'ordre d'exécution, la condition de synchronisation est naturellement respectée.

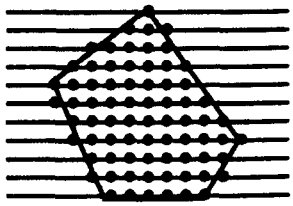


Figure 41 : Remplissage du contour (multipipeline)

{ cf ANNEXE C : REMPLISSAGE PARALLELE 3 }

Remarques :

- L'analyse est centralisée mais elle est simple, la fréquence d'envoi des commandes est donc élevée.
- La phase de remplissage peut être différée. Une scène peut être visualisée en mode filaire dans un premier temps (en posant les marques), puis tous les polygones inscrits seront remplis en une seule passe, par une même commande de remplissage entre les marques.
- Les performances varient suivant que l'on considère que le tracé du contour fait partie ou non du remplissage.

4.4.2.3. REMPLISSAGE D'UN CONTOUR PREINSCRIT

Nous proposons deux méthodes pour résoudre les singularités :

i) EXAMEN DU VOISINAGE

La scrutation du voisinage est longue, en outre elle rend la gestion du réseau plus complexe et génère des conflits d'accès. Néanmoins une solution parallèle a été développée dans [PEL 85].

ii) TABLEAU LOGIQUE

Cette approche est un peu plus intéressante dans le cadre d'une implantation cellulaire.

Les différentes étapes du remplissage sont :

- Communiquer le tableau logique aux cellules, ceci est fait une fois pour toutes,

- Déterminer le point supérieur gauche, par un balayage orienté des cellules (limité au rectangle englobant si possible),

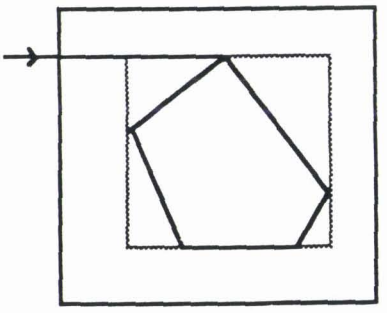


Figure 42 : Recherche du point supérieur gauche

- A partir de ce point, parcourir le contour dans le sens trigonométrique en marquant les extrémités des segments de remplissage, déterminées grâce au tableau logique. Pour ce faire, la propagation contrôlée par les données est la plus adéquate.

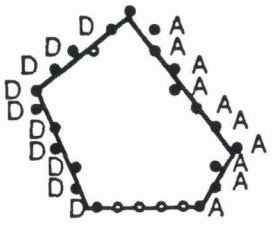


Figure 43 : Marquage des segments de remplissage

Il n'est pas indispensable de dissocier les points de départ des segments de remplissage de ceux d'arrivée. Cela pourrait permettre un contrôle au moment du remplissage.

- Envoyer sur la zone intéressée, une commande affichant les segments ainsi définis. Dans ce cas, on a recours au multipipeline.

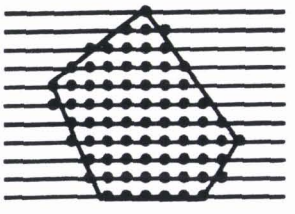


Figure 44 : Remplissage (multipipeline)

{ cf ANNEXE C : REMPLISSAGE PARALLELE 4 }

Remarques :

- Le programme permet d'effectuer le marquage à posteriori, de même la phase de remplissage peut être différée.
- La recherche du point supérieur, s'il n'est pas donné, nécessite un temps de calcul non négligeable.
- Le parcours du contour est séquentiel.
- Les communications sont multidirectionnelles et nombreuses lors du marquage.

- Pour n'utiliser qu'un type de fonctionnement, il faut choisir la propagation, ce qui nuit aux performances lors de la phase de remplissage.

4.4.3. REMPLISSAGE PAR COMPLEMENTATION

Cet algorithme dépend étroitement du tracé de segments adopté. Il peut donc être adapté aux différents types d'architectures cellulaires envisagées. Ici encore il faut analyser le contour dans son intégrité pour déterminer les points litigieux (segments horizontaux, extrémités...), on a donc recours à plusieurs types de générations de segments en fonction du nombre d'extrémités prises en compte.

Les solutions sont donc :

4.4.3.1. PROPAGATION CONTROLEE PAR LES DONNEES

L'algorithme de génération de segments est légèrement modifié : Pour chaque point particulier rencontré (cas des points de même ordonnée...), on envoie une commande de complémentation à gauche.

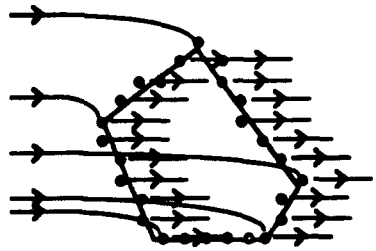


Figure 45 : Complémentation (propagation)

Le point du palier de segment à partir duquel est lancé la complémentation est moins facile à déterminer.

Ces points particuliers sont :
- ceux consécutifs à un mouvement non horizontal pour :
  - les segments orientés de haut en bas et de gauche à droite,
  - les segments orientés de bas en haut et de droite à gauche.
- ceux précédents un mouvement non horizontal pour :
  - les segments orientés de haut en bas et de droite à gauche,
  - les segments orientés de bas en haut et de gauche à droite.

(le contour étant parcouru dans le sens trigonométrique)

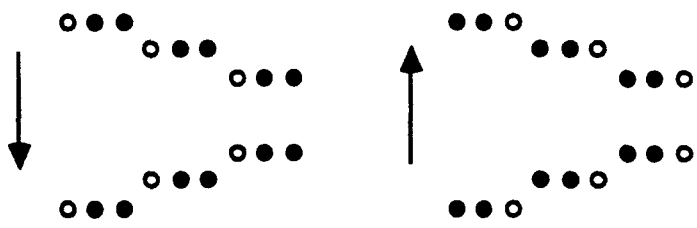


Figure 46 : Points de début de complémentation

{ cf ANNEXE C : REMPLISSAGE PARALLELE 5 }

Remarques :
- Les segments sont tracés quasiment en parallèle.
- Le contrôle du réseau est complexe, les conflits d'accès peuvent être nombreux.

4.4.3.2. MULTIPIPELINE

Remarquons que le point à partir duquel a lieu la complémentation est fonction de son orientation. Pour utiliser au mieux la solution multipipeline, il faut que toutes les commandes traversent le réseau dans le même sens.

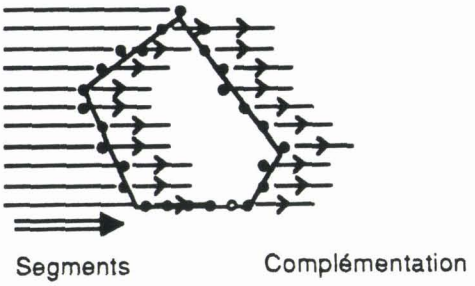


Figure 47: Complémentation (multipipeline)

La règle pour complémenter à partir d'un seul point par segment, pour une ligne de balayage donnée, est du même type que celle définie pour le marquage dans la solution du test de parité.

Le point à partir duquel à lieu la complémentation est :
- le premier point du segment rencontré sur la ligne, pour un segment orienté de haut en bas (intérieur à droite),
- le dernier point du segment rencontré sur la ligne, pour un segment orienté de bas en haut (intérieur à gauche).

{ cf ANNEXE C : REMPLISSAGE PARALLELE 6 }

Remarques :
- Les communications sont unilatérales
- Les segments sont affichés en pipeline.

Notes sur le remplissage par complémentation :

- Le prétraitement pour l'analyse globale n'est pas très long, et ne ralentit pas trop l'émission des commandes.

- Dans le contexte cellulaire, il est préférable d'utiliser une mémoire intermédiaire plutôt que de complémenter directement les pixels de l'image. Une passe supplémentaire permet ensuite la visualisation effective. Comme pour le remplissage après marquage selon le test de parité, le remplissage proprement dit peut être réduit en une passe qui consiste à afficher le pixel en fonction de la valeur contenue dans la variable intermédiaire. L'affectation de cette variable étant effectuée lors du tracé filaire.

- Pour le mode multipipeline, il est remarquable de constater que le tracé d'un polygone plein ou creux nécessite sensiblement le même temps d'exécution sur le réseau.

- Une solution SIMD pourrait être envisagée, mais elle rejoindrait l'algorithme développé à partir des équations des arêtes.

4.4.4. REMPLISSAGE A PARTIR D'UN POINT INTERIEUR

Deux types de propagations ont étés proposés lors de la présentation de l'algorithme sous sa forme séquentielle.

4.4.4.1. PROPAGATION EN LIGNES

Il est possible d'employer cette méthode pratiquement sans modification pour le réseau cellulaire. En remplaçant les accès à la pile par des communications, et les procédures par des commandes particulières.

Dans la solution séquentielle, l'examen des lignes est limité en fonction de la ligne de remplissage précédente. Cela est obtenu par le truchement de deux variables globales *xdroite* et *xgauche*.

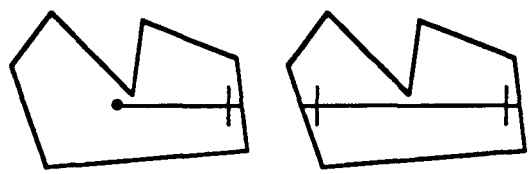


Figure 48 : Bornes pour l'exploration

Dans un réseau cellulaire, il n'y a pas de données partagées, il faut donc avoir recours à un autre stratagème :

- soit les données sont communiquées au travers du réseau en temps que paramètres de commande mais alors le trajet est doublé,

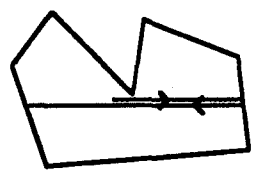


Figure 49 : Transport des bornes

- soit les cellules des lignes supérieures et inférieures sont marquées, mais cela pose un problème de synchronisation : il ne faut pas lancer un examen avant que la marque correspondante ne soit posée.

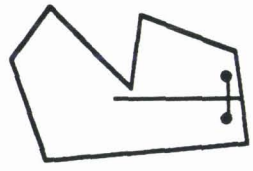


Figure 50 : Marquage des bornes

De plus, il faut différencier les marques, car plusieurs remplissages peuvent interférer.

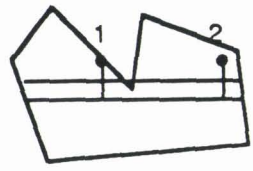


Figure 51 : Conflits entre les marques

Les marques sont donc numérotées.
Les problèmes de synchronisation ne peuvent être résolus de façon performante.

Nous optons donc pour la première solution. L'algorithme implique la définition de trois procédures :

- REMPLIR qui permet de joindre le point du contour situé à la droite de la cellule:

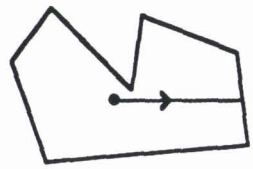


Figure 52 : Joindre le bord droit

Une fois ce point atteint, une commande de remplissage horizontal est émise.

- REMPH remplissage horizontal jusqu'à atteindre le point du contour situé à gauche.

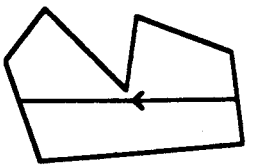


Figure 53 : Joindre le bord gauche, en remplissant

Cette commande renferme un paramètre permettant de borner l'examen demandé une fois le point du contour rejoint.

- EXAMEN parcourt la ligne en exécutant, sur chaque première cellule intérieure détectée, un remplissage.

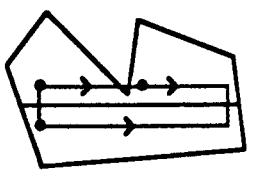


Figure 54 : Explorer les lignes voisines

{ cf ANNEXE C : REMPLISSAGE PARALLELE 7 }

Remarques :

- Cette solution, si elle génère peu de conflits d'accès, conserve un caractère séquentiel important.
- Les cellules sont activées plusieurs fois pour un même remplissage, en revanche les traitements sont très simples.
- Le comportement du réseau au cours du remplissage est assez difficile à gérer.

4.4.4.2. PROPAGATION AUX VOISINS DIRECTS

Si le principe est élémentaire, son utilisation dans la pratique pose des problèmes par le nombre élevé de transmissions inutiles. Il n'est pas possible de restreindre les directions de propagation sous peine de voir le remplissage échouer pour certaines figures.

Figure 55 : Diffusion 4 voisins 8 voisins

{ cf ANNEXE C : REMPLISSAGE PARALLELE 8 }

Remarques :

- Les nombreux conflits générés peuvent être résolus par des protocoles de synchronisation liés à l'architecture, pour ne pas ralentir l'exécution.

- La majoration du temps d'exécution est souvent très supérieure au temps réel de l'exécution. Un test d'activité sur les cellules est généralement préférable.

- Communications :

  • séquentielles : la commande est envoyée de préférence à la cellule opposée à la cellule émettrice, qui génère moins de conflits en moyenne. Un ordre de priorité est fixé pour les transmissions :

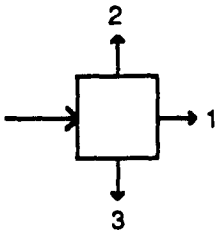


Figure 56 : Priorités des émissions

Cela ne diminue pas le nombre de conflits, mais peut accélérer l'exécution.

  • parallèles : diffusion. Les communications sont asynchrones, pour ne pas générer d'inter-blocages. Les messages en attentes sont détruits si la commande a déjà été exécutée.

4.4.5. REMARQUES D'ORDRE GENERAL

Remarquons d'abord que, plus la définition de l'image croît, plus les problèmes rencontrés sont rares proportionnellement au nombre de points total. En effet, les points confondus sont en moyenne moins nombreux :

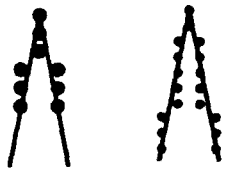


Figure 57 : Points confondus

D'autres critères, tenant compte du contexte de l'application interviennent dans le choix de la solution finale :

- Les algorithmes d'ombrage peuvent être développés sur la base de ceux de remplissage. Le choix doit alors être judicieux pour ne pas entraîner de modifications trop importantes.

  Par exemple, la solution proposée par GOURAUD [GOU 80] nécessite des interpolations linéaires. Selon la méthode adoptée celles-ci seront plus ou moins gourmandes en temps de calcul. De manière générale, les algorithmes à propagation ne sont pas compatibles avec ce genre de méthodes.

- D'autre part, si l'espace à représenter est de dimension trois, les objets manipulés sont généralement subdivisés en facettes élémentaires, afin de faciliter les divers traitement susceptibles de leur être appliqués. Dans ce cas, le prédécoupage est déjà effectué, et un algorithme simple de remplissage de surfaces élémentaires est suffisant.

4.5. TABLEAUX RECAPITULATIFS

Rappels : Numérotation des algorithmes

1 : SIMD (équation)
2 : MULTIPIPELINE (direct, test de parité)
3 : MULTIPIPELINE (marquage, test de parité)
4 : SUIVI DE CONTOUR (tableau logique, test de parité)
5 : SUIVI DE CONTOUR (complémentation)
6 : MULTIPIPELINE (complémentation)
7 : PROPAGATION (dirigée)
8 : PROPAGATION (diffusion)

Nous ne pouvons pas comparer ces solutions sans faire au préalable quelques remarques :

- Il y a deux types de problèmes : le remplissage direct des contours, le remplissage de contour préinscrits. Notons que les algorithmes capables de remplir des contours préinscrits, peuvent évidemment faire du remplissage direct, il suffit pour cela de considérer le tracé comme partie intégrante du remplissage. Il ne faut toutefois pas oublier le problème du point germe, même s'il est relativement simple lorsque les équations des arêtes sont connues.

- Pour la plupart des algorithmes les performances sur le réseau sont intimement liées à celles des algorithmes correspondant pour le segment.

- Un certain nombre de solutions sont d'ores et déjà à écarter :

  - La solution SIMD 1 placée dans le contexte de polygones quelconques, est peu attractive. Le précalcul nécessaire pour déterminer la manière d'associer les demi-plans est trop important. A cela s'ajoutent les modestes performances de l'algorithme engendrant les demi-plans (cf 2.5.)

  - Le remplissage direct par test de parité 2 est rejeté pour des raisons similaires. En fait, la quasi totalité du travail est exécutée dans la machine hôte. On rejoint dans ce cas le concept de tracé de segment de LOCEFF (cf 2.5.), pour lequel le réseau est le plus performant puisque n'ayant qu'un rôle de module de visualisation, mais que l'on est incapable d'alimenter.

  - L'algorithme utilisant un tableau logique 4 est également rejeté pour des raisons d'efficacité sur le réseau, et de complexité de gestion. En effet, il faut nécessairement une méthode à suivi de contour pour le tracé de segments, étant donné la multitude de communications diverses requises, et il faut également un multipipeline pour effectuer le remplissage par la suite.

  - La propagation dirigée 7 offre peu d'intérêt sur un réseau cellulaire, elle complexifie par trop le routage sans apporter pour cela d'accroissement de performance sensible.

Il en résulte que seules quatre hypothèses sont retenues :

MULTIPIPELINE : 3 Inscription avec marquage en fonction du test de parité
6 Par complémentation
SUIVI DE CONTOUR : 5 Par complémentation
PROPAGATION : 8 Diffusion

Remarques :

- L'architecture à PROPAGATION PURE n'apparait plus dans cette liste. Le développement d'un algorithme par complémentation ou test de parité serait envisageable, mais nécessiterait une étude spécifique complexe ne se traduisant pas par un gain substantiel sur le plan des performances.

  La PROPAGATION LIMITEE, telle que nous l'avons vue auparavant, n'est pas adaptée au problème, la limitation est plus directement liée au contour, se rapprochant ainsi de l'approche par SUIVI DE CONTOUR.

- Pour le remplissage d'un contour préinscrit la seule méthode conduisant en un temps réduit à un remplissage correct est la diffusion à partir d'un point germe, il n'y a donc pas de comparaison possible. Nous étudierons cette méthode en tant que deuxième phase d'un remplissage direct, formé d'un tracé de contour simple, puis d'un remplissage par diffusion.

Les tableaux que nous allons dresser sont relatifs aux différents algorithmes de remplissage direct.

L'étude de la méthode par diffusion suppose l'adoption d'un tracé de contour associé, nous utiliserons l'algorithme de tracé de segments par PROPAGATION LIMITEE (cf ANNEXE A algorithme N°6) car c'est celui dont le comportement est le plus approchant, ses performances sont par contre assez mauvaises.

	MULTIPIPELINE		SUIVI DE CONTOUR	PROPAGATION LIMITEE
	S	S	S	S
Cellules actives au temps (t)	H		Na x La	H x L
Nombre d'étapes successives	N		La + N	H x L
Cellules activées	H x N		Na x La x N	H x L
Communications hôte vers cellules	La x Na		Na protocoles	Na + 1 protocoles
Communications entre cellules	Droite à gauche câblées		Logiques unidirectionnelles	Câblées Multidirectionnelles
Communications cellules vers hôte	N câblées		N câblées	Time out Test d'activité

N : Le réseau est de taille NXN;
H : Hauteur du polygone;
L : Largeur du polygone;
Na : Nombre d'arêtes;
La : Longueur des arêtes;

Figure 58 : Tableau des comportements sur le réseau

L'évaluation du nombre d'étapes est directement liée à la complexité de la figure, le majorant que nous proposons est très supérieur au nombre d'étapes nécessaires pour un polygone de complexité moyenne.

La plupart des valeurs données sont des majorants, il est impossible d'évaluer de manière générale les temps nécessaires à l'affichage d'un polygone quelconque sans une étude statistique approfondie.

	MULTIPIPELINE		SUIVI DE CONTOUR	PROPAGATION LIMITEE
	3	6	5	8
Travail dans le hôte	Analyse globale du contour	Analyse globale du contour	Analyse globale du contour	
	Evaluation et envoi des commandes de tracés de segments	Evaluation et envoi des commandes de tracés de segments	Evaluation et envoi des commandes de tracés de segments avec complémentation	Evaluation et envoi des commandes de tracés de segments
				Recherche d'un point intérieur
	Envoi d'une vague de commandes pour le remplissage	Envoi d'une vague de commandes pour le remplissage		Envoi d'une commande de remplissage
Travail dans les cellules	Affichage du segment avec marques	Affichage du segment avec complémentation dans une variable intermédiaire	Affichage du segment avec complémentation directe	Affichage du segment
	Vague de remplissage	Vague de remplissage		Remplissage par diffusion

Figure 59 : Déroulement de l'exécution de la commande

Nous remarquons que pour les solutions MULTIPIPELINE 3 et 6 le temps nécessaire au remplissage des polygones est identique à celui nécessaire au tracé simple des arêtes. Le réseau sera donc très rapide, mais il sera encore plus difficile que pour le segment de l'alimenter en conséquence.

Pour la solution à SUIVI DE CONTOUR 5 les performances sont également proches de l'affichage d'un ensemble de segments quelconques, mais la complémentation engendre un parallélisme objet bien plus faible. Si nous considérons que 1% des cellules peuvent être actives simultanément, soit un parallélisme objet de 10000 segments pour un réseau 1000x1000, pour des polygones le parallélisme sera divisé par le produit Na x La_m, où Na est le nombre d'arêtes et La_m la longueur moyenne des arêtes.

Soit pour La_m = 20 ==> 500 arêtes tracées simultanément
et pour La_m = 300 ==> 3 arêtes simultanées.

Les performances à attendre de l'algorithme de remplissage par diffusion à partir d'un point germe ☠ sont très difficiles à évaluer. Plus encore que pour les autres solutions, elles sont fortement dépendantes de la complexité du polygone à remplir. Le parallélisme objet est fonction de la surface occupée par les polygones.

Nous allons évaluer les performances que l'on peut attendre des différentes méthodes, pour le tracé d'un polygone convexe.

Rappel :

N	: Taille du réseau
Nb	: Nombre de bits pour coder un entier
Op_c	: Temps pour une opération élémentaire dans une cellule
Op_h	: Temps pour une opération élémentaire dans l'ordinateur hôte
$Pt_r(i)$	: Protocole de communication dans le réseau pour i paramètres entiers
Pt_s	: Protocole de sortie cellule vers hôte (pour le contrôle)
$Th_{(i)}(j)$	: Transmission directe du hôte vers i cellules, d'une commande à j paramètres
$Tc_{(i)}(j)$	: Transmission directe d'une cellule vers i cellules, d'une commande à j paramètres
$Tc_{L(i)}(j)$	: Transmission logique d'une cellule vers i cellules, d'une commande à j paramètres

Sommets du polygone : $(S_1, \ldots S_{Ns})$

◊ Algorithme N° 3

$$T_{hote} = T_{hote}(\text{Analyse du contour}) + T_{hote}(\text{Tracé du contour}) + T_{hote}(\text{Remplissage})$$

- T_{hote}(Analyse du contour) : Pour une analyse simple telle que nous l'envisageons (cf ANNEXE C, Analyse_Globale), en ne tenant pas compte de la préparation des commandes de tracés des arêtes, le temps peut être évaluer grossièrement à :

$$T_{hote}(\text{Analyse du contour}) = Ns * (6*Op_h)$$

- T_{hote}(Tracé du contour) : Nous approchons le temps nécessaire à l'évaluation des paramètres des commandes par celui correspondant pour des tracés de segments simples (cf ANNEXE A, Algorithme N°2).

$$T_{hote}(\text{Tracé du contour}) = Ns * ((11 + 2*Nb) * Op_h + Th_{(\Delta y)}(7))$$

Remarque :
Nous verrons pourquoi il y a un paramètre supplémentaire dans la commande lors de l'étude du réseau.

- T_{hote}(Remplissage) : Il est réduit à l'émission d'une commande avec un paramètre booléen initialisé à faux, indiquant que l'on est à l'extérieur du polygone au départ).

$$T_{hote}(\text{Remplissage}) = Th_{(N)}(1)$$

$$\Longrightarrow T_{hote} = Ns * ((17 + 2*Nb) * Op_h + Th_{(\Delta y)}(7)) + Th_{(N)}(1)$$

$$T_{réseau} = T_{réseau}(\text{Tracé du contour}) + T_{réseau}(\text{Remplissage})$$

- $T_{réseau}$(Tracé du contour) : Même en supposant qu'il existe deux commandes différentes suivant l'orientation du segment (haut-bas ou bas-haut), il est nécessaire d'ajouter un paramètre et un test associé, pour ne pas poser plusieurs marques sur une même ligne.
Par analogie au tracé de segments nous obtenons :

$$T_{réseau}(1^{ère}\ \text{arête}) = N * (2*Op_c + Tc_{(1)}(7)) + (5 + Nb) * Op_c$$

Le parallélisme objet inhérent au multipipeline permet d'émettre plusieurs commandes de tracés de segments simultanément, sous réserve qu'il n'y ait pas d'interférence entre les ordonnées des points :

Dans un premier temps, nous dissocions les arêtes orientées de bas en haut de celles orientées de haut en bas.

Le fait que deux segments successifs ont une extrémité en commun nous oblige à créer de nouvelles classes en prenant un segment sur deux pour chaque classe :

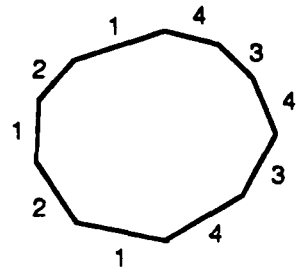


Figure 60 : Quatre ensembles d'arêtes

Remarque :
Même si une extrémité commune n'est marquée qu'une seule fois, le calcul des points du segment la prend en compte deux fois, d'où la subdivision précédente.

Le tracé d'un polygone convexe requiert donc 4 vagues de multipipeline pour le tracé du contour.

$$T_{réseau}(\text{Tracé du contour}) = (N+3) * (2*Op_c + Tc_{(1)}(7)) + (5 + Nb) * Op_c$$

- $T_{réseau}$(Remplissage) : Cette vague est lancée directement à la suite des commandes de tracé, le pipeline est donc amorcé.

$$T_{réseau}(\text{Remplissage}) = 2 * Op_c$$

$$\Longrightarrow T_{réseau} = (N+3) * (2*Op_c + Tc_{(1)}(7)) + (7 + Nb) * Op_c$$

◊ Algorithme N° 6

$$T_{hote} = T_{hote}(\text{Analyse du contour}) + T_{hote}(\text{Tracé du contour}) + T_{hote}(\text{Remplissage})$$

- T_{hote}(Analyse du contour) : L'analyse est la même que pour le test de parité.

$$T_{hote}(\text{Analyse du contour}) = Ns * (6*Op_h)$$

- T_{hote}(Tracé du contour) : Comme pour la solution précédente, nous approximons le temps nécessaire à l'évaluation des paramètres des commandes par celui correspondant pour des tracés de segments simples (cf ANNEXE A, Algorithme N°2).

$$T_{hote}(\text{Tracé du contour}) = Ns * ((11 + 2*Nb) * Op_h + Th_{(\Delta y)}(7))$$

Remarque :
Le paramètre supplémentaire n'a plus la même signification que précédemment, il peut prendre trois valeurs indiquant que :
- la cellule précédente n'appartient pas au segment,
- la cellule précédente appartient au segment,
- il faut complémenter.

- T_{hote}(Remplissage) : Le remplissage consiste simplement à afficher le pixel de la cellule en fonction d'une variable interne mise à jour par les commandes de tracé. Cette commande ne possède pas de paramètre, le travail du hôte se réduit donc à une simple transmission.

$$T_{hote}(\text{Remplissage}) = Th_{(N)}(0)$$

$$\Longrightarrow T_{hote} = Ns * ((17 + 2*Nb) * Op_h + Th_{(\Delta y)}(7)) + Th_{(N)}(0)$$

$$T_{réseau} = T_{réseau}(\text{Tracé du contour}) + T_{réseau}(\text{Remplissage})$$

- $T_{réseau}$(Tracé du contour) : Le temps nécessaire est identique à peu de chose près au temps pour l'affichage simple, il suffit de rajouter un test pour la complémentation.

$$T_{réseau}(\text{Tracé du contour}) = (N+3) * (2*Op_c + Tc_{(1)}(7)) + (5 + Nb) * Op_c$$

- $T_{réseau}$(Remplissage) : Cette vague est lancée directement à la suite des commandes de tracé, le pipeline est donc amorcé.

$$T_{réseau}(\text{Remplissage}) = Op_c$$

$$\Longrightarrow T_{réseau} = (N+3) * (2*Op_c + Tc_{(1)}(7)) + (6 + Nb) * Op_c$$

◊ Algorithme N° 5

$$T_{hote} = T_{hote}(\text{Analyse du contour}) + T_{hote}(\text{Tracé du contour})$$

- T_{hote}(Analyse du contour) : L'analyse est la même que pour les solutions précédentes.

$$T_{hote}(\text{Analyse du contour}) = Ns * (6*Op_h)$$

- T_{hote}(Tracé du contour) : Nous faisons l'évaluation en prenant comme référence l'algorithme à suivi de contour (cf ANNEXE A, Algorithme N°7).

$$T_{hote}(\text{Tracé du contour}) = Ns * (8 * Op_h + Th_{(1)}(8))$$

$$\Longrightarrow T_{hote} = Ns * (14 * Op_h + Th_{(1)}(8))$$

$$T_{réseau} = T_{réseau}(\text{Tracé du contour}) + T_{réseau}(\text{Complémentation})$$

- $T_{réseau}$(Tracé du contour) : Les performances sont rigoureusement équivalentes à celles du tracé de segments, mis à part le lancement éventuel en fin de traitement d'une commande de complémentation.

$$T_{réseau}(\text{Tracé du contour}) = Pt_r(8) + \Delta * (Tc_{L(1)}(8) + 5*Op_c)$$

(D : Longueur moyenne des arêtes)

Remarque : Le protocole de sortie n'est plus indispensable ici, il est remplacer par l'envoi de la commande de complémentation.

- $T_{réseau}$(Complémentation) : La complémentation est une commande sans paramètre. D'autre part, elle est lancée à partir d'un point du segment, or pour le protocole nous avons considéré un chemin moyen de longueur N/2 (cf 2.5.).

$$T_{réseau}(\text{Complémentation}) = N/2 * Tc_{(1)}(0)$$

$$\Longrightarrow T_{réseau} = Pt_r(8) + \Delta * (Tc_{L(1)}(8) + 5*Op_c) + N/2 * Tc_{(1)}(0)$$

◊ Algorithme N° 8

$T_{hote} = T_{hote}$(Tracé de contour)+ T_{hote}(Point germe)

- T_{hote}(Tracé du contour) : Nous avons pris par hypothèse l'algorithme de tracéde segments par propagation limitée (cf ANNEXE A, Algorithme N°6), c'est un des plus mauvais en performances théoriques, mais son comportement approche celui de l'algorithme de remplissage.

  T_{hote}(Tracé du contour) = $Ns * (10 * Op_h + Th_{(1)}(8))$

- T_{hote}(Point germe) : Le problème de trouver un point intérieur est très simple pour le cas particulier des polygones convexes. Il suffit de prendre le barycentre de deux points non consécutifs : cela nécessite donc 2 additions et 2 divisions par deux (des divisions entières qui plus est).

  T_{hote}(Point germe) = $4 * Op_h$

==> $T_{hote} = Ns * (10 * Op_h + Th_{(1)}(8)) + 4 * Op_h$

$T_{réseau}$ = $T_{réseau}$(Tracé du contour) + $T_{réseau}$(Remplissage)

- $T_{réseau}$(Tracé du contour) : Les segments sont tracés en parallèle.

  $T_{réseau}$(Tracé du contour) = $Pt_r(8) + \Delta * (Tc_{(4)}(8) + 6*Op_c) + Op_c$

- $T_{réseau}$(Remplissage) : Nous donnons un majorant, on considère le cas extrême du remplissage du rectangle englobant, avec comme point germe un de ses coins.

  $T_{réseau}$(Remplissage) = $Pt_r(0) + (\Delta x_p + \Delta y_p) * (Op_c + Tc_{(4)}(0))$
  (Δx_p = Différences entre les abscisses minimales et maximales du polygone, Δy_p = Différences entre les ordonnées minimales et maximales du polygone)

==> $T_{réseau} = Pt_r(8) + Pt_r(0) + \Delta * (Tc_{(4)}(8) + 6*Op_c) + Op_c + (\Delta x_p + \Delta y_p) * (Op_c + Tc_{(4)}(0))$

A partir de ces quelques éléments nous pouvons faire quelques remarques sur les solutions proposées.

Il y a trois architectures possibles :

- MULTIPIPELINE : Les performances des algorithmes 3 et 6 sont comparables, avec néanmoins un léger avantage pour la méthode par complémentation.
- SUIVI DE CONTOUR : Algorithme N°5
- PROPAGATION LIMITEE : Algorithme N°8 C'est la seule architecture pouvant traiter de façon intéressante le remplissage de contours préinscrits.

Pour l'affichage de polygones pleins, à partir de la donnée de leurs arêtes, nous remarquons que les algorithmes 3, 5 et 6 fournissent de bonnes performances mais en sollicitant fortement la machine hôte. En revanche, la solution 8 est moins coûteuse en temps machine pour la préparation de commandes, mais c'est au détriment des performances du réseau (cela est dû pour une bonne part au tracé de contour associé qui est peu performant).

Si les performances des solutions multipipelines sont liées au nombre d'arêtes des polygones et peuvent être déduites de l'étude du tracé de segments, il n'en est pas de même pour le suivi de contour.
En effet, le taux de parallélisme espéré chute sensiblement à cause des commandes de complémentation émises qui occupent fortement le réseau.

Pour donner quelques chiffres significatifs permettant de comparer les performances de nos machines avec celles d'autres solutions proposées dans la littérature, nous allons traiter le cas particulier du remplissage de triangles.

Ce problème est souvent pris comme référence car bon nombre d'applications utilisent la triangulation pour modéliser des surfaces complexes.

Il est généralement admis, pour les mesures de performances, que les triangles ont une hauteur moyenne de l'ordre d'une vingtaine de pixels.

Comme pour le tracé de segments ou de cercles, nous nous intéressons principalement aux performances obtenues sur le réseau.

◊ Algorithme N° 3

Le tracé d'un triangle plein nécessite deux vagues de multipipeline, une contenant une arête, l'autre les deux restantes. Le partage s'effectue par rapports aux sommets supérieur et inférieur du triangle. Le fait de n'avoir à traiter que des triangles, nous permet de ne pas tenir compte de ces remarques. Une construction judicieuse des vagues de commandes masquera cette répartition.

Si de plus, nous négligeons la vague supplèmentaire nécessaire à l'affichage des pixels (en fonction de la variable mise à jour pendant le tracé), le temps requis pour la tracé des triangles est identique à celui du simple tracé des arêtes.

⇒ Débit = 32 Millions de triangles par seconde

(i.e. le tiers des performances du tracé de segments)

Remarque :

Le nombre d'instructions du hôte nécessaires pour le tracé d'un triangle est trois fois supérieur à celui requis pour un segment, soit 90 instructions.

◊ Algorithme N° 6

Les remarques faites sur l'algorithme précédent s'appliquent sans restriction au cas présent :

⇒ Débit = 83 Millions de triangles par seconde

Remarque :

Nombre d'instructions par triangle = 186

◊ Algorithme N° 5

Toujours en nous référant aux études menées pour le tracé de segments, nous estimons que le temps sera quarante fois supérieur pour tracer un triangle. En effet, le problème n'est pas dû au temps nécessaire à l'affichage d'un triangle, légèrement supérieur à celui estimé pour le tracé de ses arêtes (le surplus est à attribuer au remplacement du protocole de sortie par l'émission de la commande de complémentation).

Le rapport de performances est dû au taux d'occupation du réseau. En effet, alors qu'il y avait une cellule active tout au long du traitement, il y en a désormais une par groupe de points de même ordonnée pour une arête.

⇒ Débit = 2,5 Millions de triangles par seconde

Remarque :

Nombre d'instructions par triangle = 186

◊ Algorithme N° 8

Cette solution n'est pas adaptée au problème traité.

Néanmoins nous allons effectuer une estimation, en faisant abstraction du tracé proprement dit.

En se fondant sur les remarques faites pour le tracé de segments (cf 2.5.) nous pouvons donner le nombre approximatif de triangles pouvant être tracés simultanément :

$$Nm = \frac{5}{8}\frac{N^2}{\Delta^2} = 400 \text{ triangles simultanés}$$

Temps pour un triangle = 10000 + 40 * (20 + 20) = 11,6 µs

⇒ Débit = 400/11,6 = 34 Millions de triangles par secondes

Remarque :

Le travail de la machine hôte se réduit au calcul d'un point intérieur au triangle : ((((Xa+Xb)DIV2)+Xc)DIV2),(((Ya+Yb)DIV2)+Yc)DIV2)), par exemple, soit 8 Op_h par triangle.

Rappel :

Il ne s'agit que du remplissage, à ce temps vient s'ajouter le tracé du contour aux performances modestes (cf 2.5.).

Pour cet exemple, nous remarquons que les solutions multipipelines sont celles qui offrent les meilleurs résultats. Cependant, il ne faut pas oublier que cela est obtenu au prix de l'exécution de nombreux calculs pour la machine hôte.

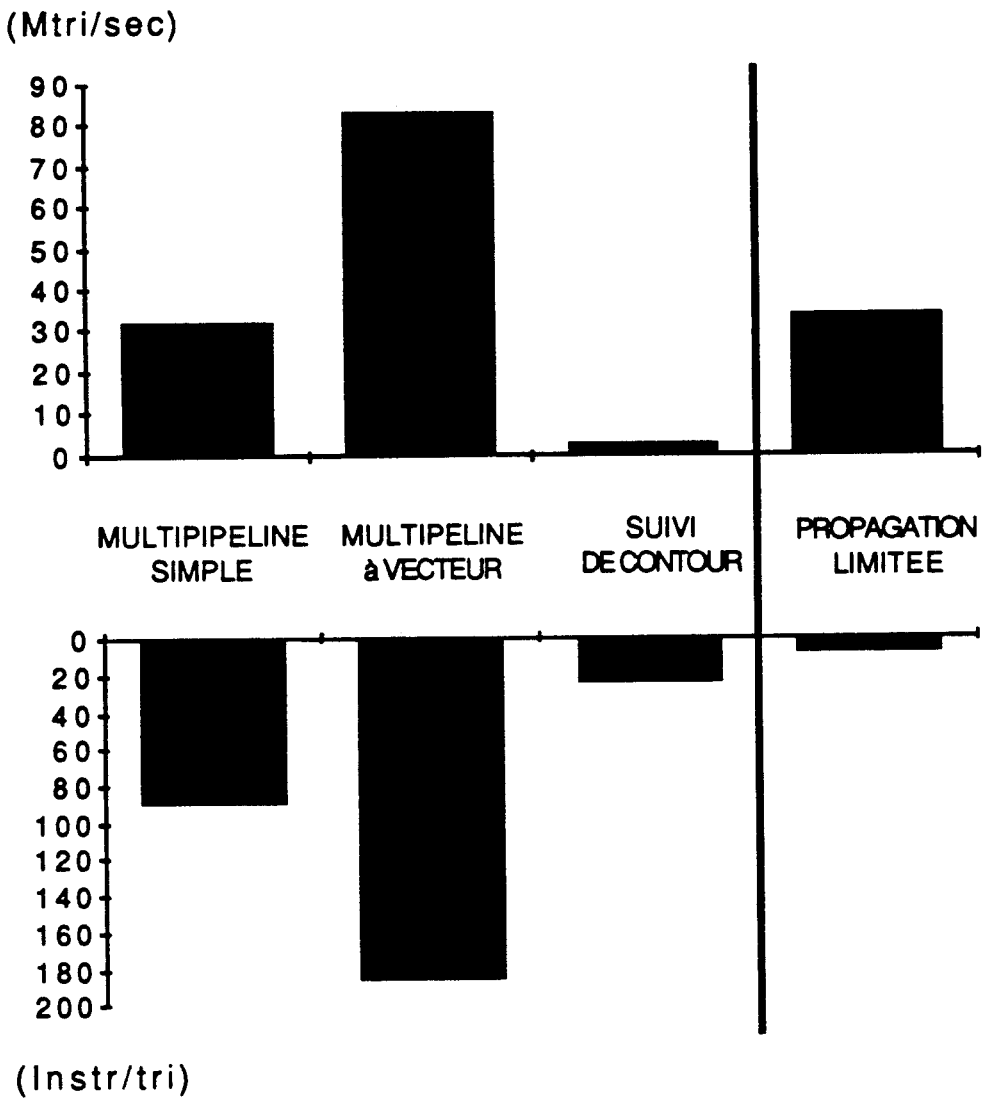
Par contre, l'utilisation du suivi de contour offre des performances beaucoup plus modestes mais en ne nécessitant que peu de précalculs.

Enfin, si l'on fait abstraction du tracé du contour, la propagation limitée est une solution performante qui ne submerge pas la machine hôte de calculs.

Il faut toutefois remarquer que les chiffres obtenus sont comme à l'accoutumée fortement liés au problème traité, et que de légères modifications de l'énoncé de celui-ci ne sont pas sans conséquences sur cette estimation.

Algorithme	Performances du hôte souhaitable (en Mips)	Performances du réseau attendues (en Mtri/secondes)	Nombre d'instructions par triangle
Multipipeline Simple	2800	32	90
Multipipeline à Vecteur	15500	83	186
Suivi de Contour	60	2,5	24
Propagation Limitée	270	34	8

PERFORMANCES DU RESEAU



NOMBRE D'INSTRUCTIONS PAR TRIANGLE

BIBLIOGRAPHIE :

[ACK 81], [BRF 79], [CHA 84], [DUN 83], [FOM 84], [FUC 85], [GOU 80], [GOP 88], [HEG 85], [HER 83], [ISO 73], [KIL 87], [LAN 83], [LIT 79], [LUC 77], [LUC 82], [MAR 82], [MER 84], [NWS 79], [OUL 88], [PAV 82], [PER 88], [PHO 75], [SCH 78], [SMI 79], [TAN 88], [THP 88], [WOO 85].

5. PERSPECTIVES DE DEVELOPPEMENT

LES PERSPECTIVES ARCHITECTURALES

Les études menées dans les pages qui précédent nous on conduit à envisager plusieurs types d'architectures :

- ◊ S.I.M.D.
- ◊ MULTIPIPELINE SIMPLE
- ◊ MULTIPIPELINE A VECTEUR D'ENTREES
- ◊ PROPAGATION PURE
- ◊ PROPAGATION LIMITEE
- ◊ SUIVI DE CONTOUR

Les solutions les plus intéressantes doivent être telles que tous les problèmes évoqués puissent être résolus de manière efficace avec un même type de fonctionnement.

D'ores et déjà, la solution S.I.M.D. est éliminée. Outre le fait que pour le tracé de segments et de cercles ses performances sont pour le moins modestes, il n'y a pas de remplissage mettant à profit l'aspect massivement parallèle de cette approche. Ce faible rendement est dû en partie à l'absence de parallélisme lié à l'émission séquentielle des commandes.

D'autre part, il subsiste un problème sous-jacent à la réalisation physique du réseau : celui de la distribution des commandes au réseau. Les solutions de diffusion simultanée sont également les plus coûteuses et relèvent des technologies d'avant-garde. D'autres hypothèses plus classiques imposent de fournir un outil pour la synchronisation du réseau, problème non trivial pour un réseau 1000x1000.

De même l'architecture à PROPAGATION PURE n'est pas retenue. Il s'est avéré que le développement d'algorithmes spécifiques à cette architecture n'offre pas d'intérêt. L'algorithmique est en effet plus complexe, il faut reprendre les problèmes à la base (notamment pour le remplissage), mais elle n'apporte pas de gain de performances sensible.

En fait, le comportement est à rapprocher de celui d'un réseau multipipeline avec pour seuls avantages de réduire de moitié le temps nécessaire à l'amorçage du réseau et de n'utiliser qu'un seul point d'entrée (la cellule centrale). Les inconvénients sont non seulement d'ordre algorithmique, mais également architecturaux.

En effet, l'existence de différents types de cellules selon leur position dans le réseau, rend l'implantation dans le silicium plus délicate. La 'tranchabilité', c'est à dire la possibilité de diviser le réseau en plusieurs entités distinctes, en souffre également et par voie de conséquence la faculté d'adaptation (accroissement de la taille du réseau par adjonction d'éléments modulaires).

La PROPAGATION LIMITEE n'est viable que si le remplissage de contours préinscrits est indispensable, car sinon les autres solutions lui seront préférées. En effet, les performances obtenues pour le tracé de segments et de cercles ne soutiennent pas la comparaison avec les autres propositions.

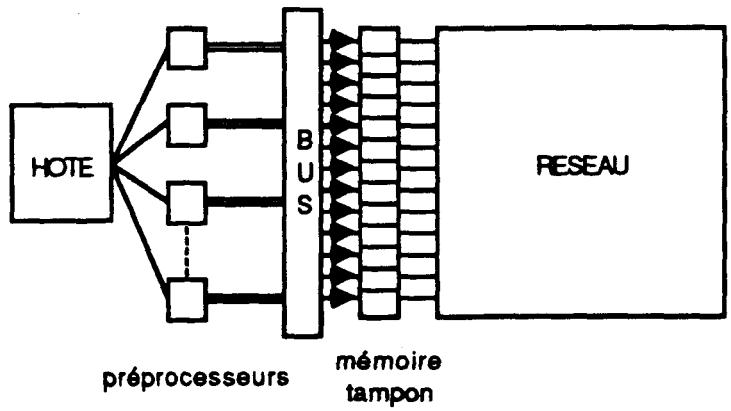
De plus, nous n'avons pas approfondi le problème du test d'arrêt. Il n'est pas possible de récupérer les commandes après leur exécution comme c'est le cas généralement. L'hypothèse du test d'activité suppose, pour qu'il soit efficace, un câblage particulier afin d'effectuer une séquence question-réponse dans les meilleurs délais sans altérer les performances des traitements en cours. La deuxième hypothèse, celle du dépassement de temps, ne convient pour le remplissage qu'au prix d'une majoration très importante du temps nécessaire.

Il reste donc trois types d'architectures auxquelles nous accordons plus d'importance.

Nous allons, en premier lieu, aborder le cas des machines MULTIPIPELINE pour signaler que le partage d'un même front entre plusieurs commandes est plus difficile qu'il n'y paraît de prime abord.

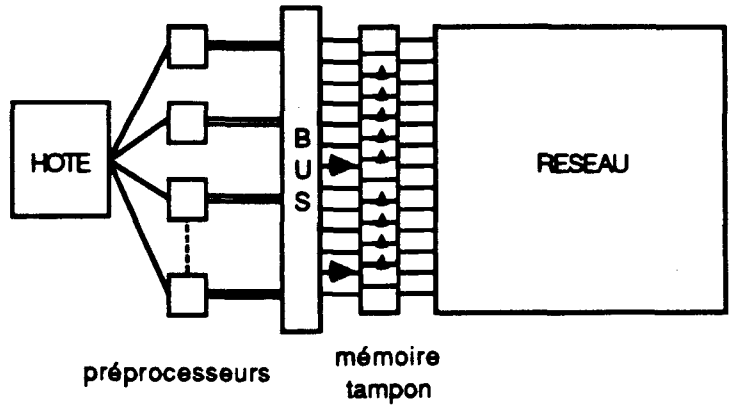
Outre le problème de la répartition des traitements dans le temps afin d'optimiser le remplissage des fronts de commandes, il faut considérer la réalisation physique du réseau.

Ne perdons pas de vue par ailleurs, que les vagues doivent se succéder à un rythme très élevé afin de ne pas altérer les performances du réseau. Dans l'état actuel de la technologie, seule la multiplication des processeurs de prétraitement est susceptible de répondre à nos besoins. Ceci sous-entend un partage de l'accès au réseau. Une solution est dans l'utilisation d'un bus rapide partagé entre les différents processeurs, et une mémoire tampon permettant la construction de la vague de commandes.

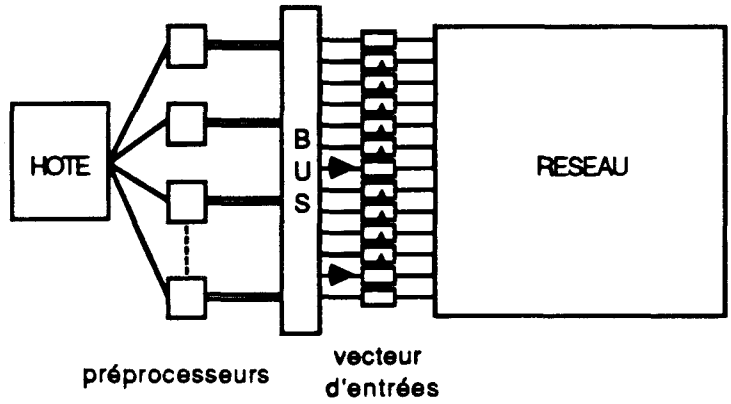


Une fois que la vague est complète, elle est émise à destination des cellules du bord du réseau.

Le problème réside dans le nombre d'accès au bus nécessaires au chargement de la mémoire tampon. Il est résolu en introduisant un peu d'intelligence dans la mémoire. Chacune des cases sera capable de savoir si la case directement supérieure doit contenir la commande et, dans l'affirmative, de lui communiquer la dite commande. Le nombre de points d'entrées dans le bus n'est pas réduit, mais il n'y a plus qu'un seul accès par commande.



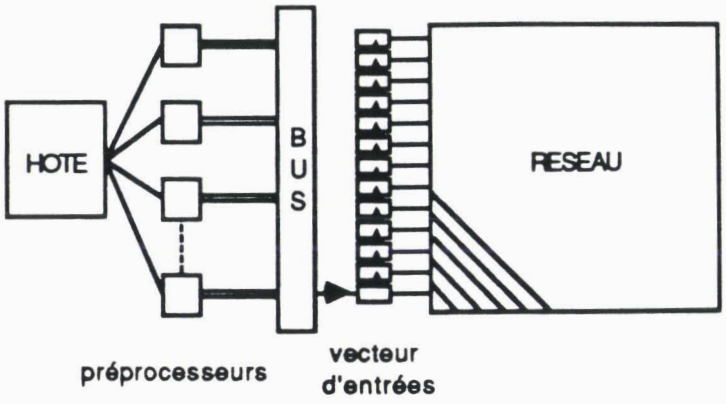
En regardant le schéma précédent nous voyons clairement apparaître la solution adoptée pour le MULTIPIPELINE A VECTEUR D'ENTREES. En effet, les calculs nécessaires à l'évaluation des paramètres dépendants de la ligne, pour une même commande, peuvent être réalisés de façon pipeline. Nous utiliserons donc, pour le tampon, des cellules capables d'effectuer quelques calculs élémentaires (une opération élémentaire pour le segment et le remplissage, 6 pour le cercle).



Remarque :

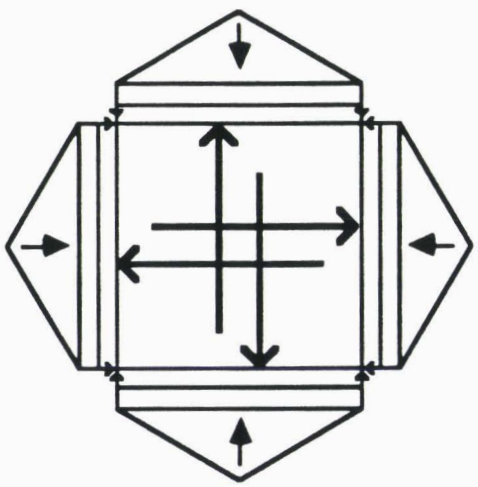
Le nombre de calculs à effectuer est du même ordre de grandeur que celui de ceux nécessaires au traitement dans les cellules du réseau.

Cette particularité peut être exploitée pour concevoir une machine plus simple avec une seule commande par vague. Dans ce cas, il devient intéressant d'effectuer une émission directe des cellules du tampon, vers les cellules du réseau sans attendre un top de synchronisation. Le temps de cycle est respecté par le pipeline de préparation des paramètres qui prend effet sur toute la largeur du réseau, simplifiant ainsi quelque peu le prétraitement effectué par ailleurs.



Remarque :

Il existe une façon très simple de multiplier par quatre les performances des réseaux multipipelines. Il suffit pour cela de superposer quatre réseaux ayant chacun leur propre direction de propagation NORD, EST, SUD, OUEST.



Chaque cellule est divisée en quatre couches indépendantes possédant chacune ses propre liens avec ses voisines, seule est partagée la variable représentant le pixel associé.

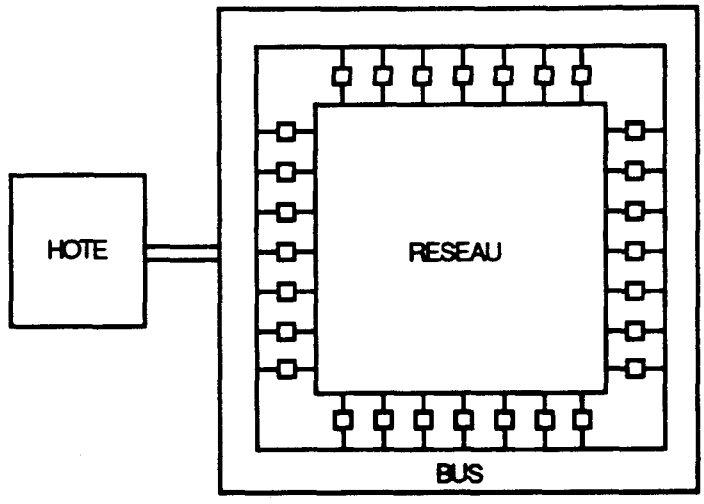
Cette solution offre l'avantage de préserver la simplicité de fonctionnement du multipipeline, certes au prix d'un développement architectural plus important, mais pas démesuré.

En revanche, la distribution des commandes sera plus délicate dans le cas du remplissage notamment, qui ne peut être réparti sur les quatres bord.

Enfin, dernière proposition : le SUIVI DE CONTOUR. D'un point de vue théorique cette solution offre des performances exceptionnelles, si abstraction est faite des nombreux conflits d'accès qui sont générés.

Aucune étude n'étant menée sur le taux d'occupation du réseau acceptable, il nous est difficile de fournir sérieusement une estimation précise des performances. Peut-il y avoir 1000, 10000, voir 100000 cellules actives simultanément sur un réseau 1000x1000 sans qu'il y ait interblocage? En fait, cela dépend des traitements à effectuer mais nécessite dans tous les cas un protocole de routage à la fois rapide et sûr capable de suspendre des traitements pour laisser d'autres se terminer afin d'éviter les interblocages.

Par contre, il est intéressant de constater que la multiplication des processeurs, dans certaines limites, ne pose pas de problème d'accès au réseau. Ce fait allié au peu de calculs nécessaires à l'évaluation des paramètres des commandes, doit nous permettre d'alimenter massivement le réseau, sous réserve de la non saturation de ce dernier.



Pour un réseau 1000x1000, 3996 processeurs de prétraitement peuvent être installés sur les bords du réseau sans conflit d'accès. La machine hôte se contente de distribuer les données, laissant le soin aux processeurs de prétraitement de construire les commandes.

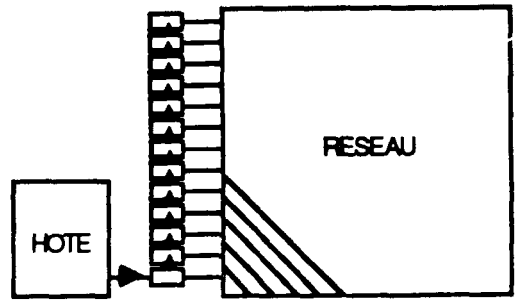
Remarque :

Cette solution revient à disposer sur les bords du réseau des cellules plus complexes capables de préparer les commandes.

LA SIMULATION

Les simulations entreprises par ATAMENIA sur un réseau de 16 TRANSPUTERS en OCCAM 2 ont permis de vérifier la validité des solutions proposées. En outre, les mesures qui ont pu être établies sont venues confirmer les évaluations théoriques. Il faut toutefois souligner qu'il s'agit d'une simulation avec des moyens peu appropriés, et que certaines performances n'ont pu être mesurées.

Une réalisation 'grandeur nature' d'un réseau simple est envisagée. Il s'agit d'un réseau multipipeline à vecteur d'entrées muni d'un seul processeur de prétraitement, dans un premier temps.



Notre choix s'est porté sur cette solution car elle offre de bonnes performances tout en conservant une certaine simplicité de fonctionnement.

DEVELOPPEMENTS ALGORITHMIQUES

Pour les algorithmes de base que nous avons présenté, la complexité de la cellule est évidemment très limitée. Mais, une étude, menée en marge par ATAMENIA, sur les problèmes d'interpolations linéaires nécessaires aux modèles d'ombrage de GOURAUD ou PHONG nous a montré que les cellules requises n'étaient pas foncièrement plus compliquées.

D'ailleurs, la généralisation à la troisième dimension des solutions proposées ne pose pas de problèmes insurmontables. En outre, nous noterons que la réalisation d'un algorithme de Z-BUFFER est particulièrement triviale sur de telles architectures.

En revanche, la manière dont on pourrait appliquer efficacement l'algorithme de lancer de rayon sur une architecture cellulaire reste un problème en suspens.

CONCLUSION GENERALE

L'utilisation des machines cellulaires pour la synthèse d'images permet d'obtenir des performances remarquables. Pour développer une architecture cohérente il faut tenir compte des spécificités des algorithmes mis en jeu.

Notre étude, portant sur les principaux algorithmes de base, montre que deux types d'architectures peuvent être envisagées avec sérieux : le multipipeline et le suivi de contour.

Le multipipeline ne pose pas de problème pour les algorithmes étudiés, mais son fonctionnement particulier, avec le faible nombre de liaisons existantes, ne se satisfait pas de n'importe quelle application.

Le suivi de contour, de fonctionnement beaucoup plus souple, soulève d'ores et déjà des problèmes de conflits d'accès et de saturation du réseau s'il est fortement sollicité. Par contre, il s'adaptera plus simplement à d'autres problèmes, mais pas nécessairement de manière performante.

La réalisation d'une machine de visualisation simple sans lancer de rayon, mais avec le modèle d'ombrage de GOURAUD devrait nous permettre de démontrer le bien-fondé de nos préférences pour l'architecture multipipeline.

ANNEXE A

LES ALGORITHMES DE TRACE DE SEGMENTS

REMARQUES PRELIMINAIRES RELATIVES AUX ALGORITHMES DE TRACES DE SEGMENTS

Nous présentons en premier lieu les algorithmes séquentiels qui ont servi de base au développement des solutions parallèles proposées à la suite.

L'objet de ces algorithmes est de tracer un segment (XA,YA) --> (XB,YB), les coordonnées sont des données globales supposées connues de l'ordinateur hôte (où sont également implémentées les solutions séquentielles).

Les points de l'image et les cellules sont repérés grâce à un même repère orthogonal dont l'origine se situe dans le coin inférieur droit (de l'image où du réseau).

Chaque cellule est supposée connaître ses propres coordonnées (I,J), ce qui n'est pas indispensable pour toutes les solutions présentées.

La procédure SEGMENTX (I,J) correspond au programme exécuté une fois que la cellule a analysé la première partie du message reçu, à savoir le nom de la commande, en l'occurence SEGMENTX.

Rappel des notations utilisées :

- Envoi à (x,y) de (commande paramétrée) : Emission d'une commande depuis l'ordinateur hôte à destination de la cellule (x,y).

- Reception (paramètres de commandes) : Reception par la cellule des paramètres associés à la commande qu'elle vient d'interpréter.

- Transmission (i,j) de (commande paramétrée) : Transmission d'une commande, d'une cellule vers une de ses voisines directes(i et j prennent leur valeur parmi -1,0,1).

  Remarque :
  Pour les solutions multipipelines, nous conservons cette même notation par souci d'homogénéité, mais il est évident que les paramètres i et j sont superflus, puisque toutes les transmissions sont identiques : Transmission (1,0) de (commande) par exemple.

- Transmission_distante (x,y) de (commande) : Transmission entre cellules non directement voisines => (x,y) sont des coordonnées exprimées dans le repère du réseau.
  Cette transmission est utilisée pour la solution dichotomique.

- Afficher sans paramètre : est utilisée dans les procédures exécutées par les cellules pour allumer le pixel qui leur est associé.

- Afficher (x,y) : est utilisée dans les solutions séquentielles pour afficher le pixel (x,y).

Remarque importante :
Les boucles POUR présentes dans les procédures de l'ordinateur hôte ne sont là que pour faciliter la compréhension. Il ne faut pas prendre en compte le caractère séquentiel qu'elles renferment. Elles correspondent dans la pratique à une diffusion *'quasiment simultanée'* d'une commande à tout ou partie du réseau. Il n'est pas du ressort de cette thèse d'étudier la manière dont l'ordinateur hôte transmet physiquement ses commandes au réseau.

Fonctions mathématiques :

- INF (x,y) : retourne la plus petite des deux valeurs x et y.

- SUP (x,y) : retourne la plus grande des deux valeurs x et y.

- ABS (x) : retourne la valeur absolue de x.

- SGN (x) : retourne 1 si l'argument est positif, 0 si l'argument est nul, -1 sinon.

- ARRONDI (x) : retourne l'entier le plus proche du réel x.

- DIV : opérateur de division entière. Rappelons d'ailleurs que (DIV 2) et (* 2) correspondent à de simples décalages et ne posent pas de problèmes pour le câblage éventuel.

- MAXINT : entier le plus grand.

ALGORITHME DE TRACE DE SEGMENTS SEQUENTIEL N°1

BRESENHAM

```
PROCEDURE SEGMENT1 ;

  DEBUT
     difx := XB -XA ;
     dify := YB - YA ;
     diax := SGN (difx) ;
     diay := SGN (dify) ;
     max := ABS (difx) ;
     min := ABS (dify) ;
     SI  (max <= min) ALORS
        DEBUT
           echange := min ;
           min := max ;
           max := echange ;
           axix := 0 ;
           axiy := diay ;
        FIN
     SINON
        DEBUT
           axix := diax ;
           axiy := 0 ;
        FIN ;
     x := XA ;
     y := YA ;
     seuil := max DIV 2 ;
     POUR I :=0 A max FAIRE
        DEBUT
           Afficher  (x,y)  ;
           seuil := seuil + min ;
           SI  (seuil >= max)  ALORS
              DEBUT
                 seuil := seuil - max ;
                 x := x + diax ;
                 y := y + diay ;
              FIN
           SINON
              x := x + axix ;
              y := y + axiy ;
        FIN
  FIN ;
```

ALGORITHME DE TRACE DE SEGMENTS SEQUENTIEL N°2

D.D.A.

```
PROCEDURE SEGMENT2 ;

  DEBUT
     difx := XB - XA ;
     dify := YB - YA ;
     adifx := ABS (difx) ;
     adify := ABS (dify) ;
     SI  (adifx > adify)  ALORS
        longueur := adifx ;
     SINON
        longueur := adify ;
     xincrement := difx / longueur ;
     yincrement := dify / longueur ;
     x := XA ;
     y := YA ;
     POUR  compteur:=0  A  longueur  FAIRE
        DEBUT
           Afficher (ARRONDI (x), ARRONDI (y));
           x := x + xincrement ;
           y := y + yincrement ;
        FIN
  FIN ;
```

ALGORITHME DE TRACE DE SEGMENTS SEQUENTIEL N°3

DICHOTOMIQUE

```
PROCEDURE SEGMENT3 ;

  DEBUT
    TRACER3 (XA, YA, XB, YB) ;
  FIN

PROCEDURE TRACER3 (x1, y1, x2, y2) ;

  DEBUT
    SI ((x1<>x2) ET (y1<>y2)) ALORS
       DEBUT
          xm := (x1+y1) / 2 ;
          ym := (x2+y2) / 2 ;
          Afficher (ARRONDI(xm), ARRONDI(ym)) ;
          TRACER3 ((x1, y1),(xm, ym)) ;
          TRACER3 ((xm, ym),(x2, y2)) ;
       FIN
  FIN ;
```

ALGORITHME DE TRACE DE SEGMENTS PARALLELE N°1

S.I.M.D.

```
PROCEDURE SEGMENT1_HOTE ;

  DEBUT
     xmin := INF (XA,XB) ;
     ymin := INF (YA,YB) ;
     xmax := SUP (XA,XB) ;
     ymax := SUP (YA,YB) ;
     difx := XA - XB ;
     dify := YA - YB ;
     seuil := SUP (ABS (difx),ABS (dify)) DIV 2 ;
     constante := dify*XA - difx*YA ;
     POUR I := 0 A Taille_reseau FAIRE
        POUR J := 0 A Taille_reseau FAIRE
           Envoi à (i,j) de (SEGMENT1, xmin, ymin, xmax, ymax, difx, dify, seuil,
                             constante) ;
  FIN ;


PROCEDURE SEGMENT1_CELLULE (I,J) ;

  DEBUT
     Reception (xmin, ymin, xmax, ymax, difx, dify, seuil, constante) ;
     SI ((xmin<= I) ET (I<= xmax) ET (ymin<=J) ET (J<= ymax)
     ET (ABS (-dify*I+difx*J + constante) <= seuil))  ALORS
        Afficher;
  FIN ;
```

ALGORITHME DE TRACE DE SEGMENTS PARALLELE N°2

MULTIPIPELINE

```
PROCEDURE SEGMENT2_HOTE ;

  DEBUT
     xmin := INF (XA,XB) ;
     ymin := INF (YA,YB) ;
     xmax := SUP (XA,XB) ;
     ymax := SUP (YA,YB) ;
     difx := XA - XB ;
     dify := YA - YB ;
     seuil := SUP (ABS (difx),ABS (dify)) DIV 2 ;
     erreur := dify*XA - difx*YA ;
     POUR j := ymin A ymax FAIRE
        Envoi à (0,j) de (SEGMENT2, xmin, xmax, difx, dify, seuil, erreur) ;
  FIN ;


PROCEDURE SEGMENT2_CELLULE (I,J) ;

  DEBUT
     Reception (xmin, xmax, difx, dify, seuil, erreur) ;
     Transmission (1,0) de (SEGMENT2, xmin, xmax, difx, dify, seuil, erreur-dify) ;
     SI ((xmin<= I) ET (I<= xmax) ET (ABS (erreur + difx*J) <= seuil)) ALORS
        Afficher;
  FIN ;
```

ALGORITHME DE TRACE DE SEGMENTS PARALLELE N°3

MULTIPIPELINE A VECTEUR

```
PROCEDURE SEGMENT3_HOTE ;

  DEBUT
     xmin := INF (XA,XB) ;
     ymin := INF (YA,YB) ;
     xmax := SUP (XA,XB) ;
     ymax := SUP (YA,YB) ;
     difx := XA - XB ;
     dify := YA - YB ;
     seuil := SUP (ABS (difx),ABS (dify)) DIV 2 ;
     constante := dify*XA - difx*YA ;
     erreur := difx * ymin + constante ;
     POUR j := ymin A ymax FAIRE
        DEBUT
           Envoi à (0,j) de (SEGMENT3, xmin, xmax, dify, seuil, erreur) ;
           erreur := erreur + difx ;
        FIN
  FIN ;


PROCEDURE SEGMENT3_CELLULE (I,J) ;

  DEBUT
     Reception (xmin, xmax, dify, seuil, erreur) ;
     Transmission (1,0) de (SEGMENT3, xmin, xmax, dify, seuil, erreur-dify) ;
     SI ((xmin<= I) ET (I<= xmax) ET (ABS (erreur)<= seuil)) ALORS
        Afficher;
  FIN ;
```

ALGORITHME DE TRACE DE SEGMENTS PARALLELE N°4

MULTIPIPELINE FRACTIONNAIRE

```
PROCEDURE SEGMENT4_HOTE ;

  DEBUT
    xmin := INF (XA,XB) ;
    ymin := INF (YA,YB) ;
    xmax := SUP (XA,XB) ;
    ymax := SUP (YA,YB) ;
    difx := XA - XB ;
    dify := YA - YB ;
    SI (dify=0) ALORS
      Envoi à (0,ymin) de (SEGMENT4, xmin, xmax, MAXINT, 0)
    SINON
      DEBUT
      A := difx/dify ;
      B := XA - A*YA ;
      seuil := SUP(ABS (A),1) ;
      erreur := A*ymin + B ;
      POUR j := ymin A ymax FAIRE
        DEBUT
          Envoi à (0,j) de (SEGMENT4, xmin, xmax, seuil, 2*erreur) ;
          erreur := erreur + A ;
        FIN
      FIN
  FIN ;


PROCEDURE SEGMENT4_CELLULE (I,J) ;

  DEBUT
    Reception (xmin, xmax, seuil, erreur) ;
    Transmission (1,0) de (SEGMENT4, xmin, xmax, seuil, erreur-2) ;
    SI ((xmin<= I) ET (I<= xmax) ET (ABS (erreur)<= seuil)) ALORS
      Afficher;
  FIN ;
```

Remarque :

Les segments horizontaux requièrent un traitement particulier.

ALGORITHME DE TRACE DE SEGMENTS PARALLELE N°5

PROPAGATION PURE

```
PROCEDURE SEGMENT5_HOTE ;

  DEBUT
    xmin := INF (XA,XB) ;
    ymin := INF (YA,YB) ;
    xmax := SUP (XA,XB) ;
    ymax := SUP (YA,YB) ;
    difx := XA - XB ;
    dify := YA - YB ;
    seuil := SUP (ABS (difx),ABS (dify)) DIV 2 ;
    erreur := dify*XA - difx*YA + (Taille_reseau DIV 2) * (-dify+difx) ;
    Envoi à ((Taille_reseau DIV 2),(Taille_reseau DIV 2)) de
                    (SEGMENT5, xmin, ymin, xmax,ymax, difx, dify, seuil, erreur) ;
  FIN ;


PROCEDURE SEGMENT5_CELLULE (I,J) ;

  DEBUT
    SI NON Marque ALORS
      DEBUT
        Reception (xmin, ymin, xmax, ymax, difx, dify, seuil, erreur) ;
        Transmission (1,0) de (SEGMENT5, xmin, ymin, xmax, ymax, difx, dify, seuil,
                               erreur-dify) ;
        Transmission (-1,0) de (SEGMENT5, xmin, ymin, xmax, ymax, difx, dify,
                               seuil, erreur+dify) ;
        Transmission (0,1) de (SEGMENT5, xmin, ymin, xmax, ymax, difx, dify, seuil,
                               erreur+difx) ;
        Transmission (0,-1) de (SEGMENT5, xmin, ymin, xmax, ymax, difx, dify,
                              seuil,erreur-difx) ;
        SI ((xmin<= I) ET (I<= xmax) ET (ymin<=J) ET (J<= ymax)
        ET (ABS (erreur)<= seuil)) ALORS
          Afficher;
        Marquer;
      FIN
  FIN ;
```

Remarque :

L'algorithme présenté n'utilise pas de câblage particulier. Pour les solutions utilisant un guidage physique (cf 2.4.4.1.), l'incrémentation du paramètre est fonction du port de réception. D'autre part le marquage n'est plus alors indispensable. Ces modifications ne pertubent pas l'évaluation des performances.

ALGORITHME DE TRACE DE SEGMENTS PARALLELE N°6

PROPAGATION LIMITEE

```
PROCEDURE SEGMENT6_HOTE ;

   DEBUT
      xmin := INF (XA,XB) ;
      ymin := INF (YA,YB) ;
      xmax := SUP (XA,XB) ;
      ymax := SUP (YA,YB) ;
      difx := XA - XB ;
      dify := YA - YB ;
      seuil := SUP (ABS (difx),ABS (dify)) DIV 2 ;
      erreur := 0 ;
      Envoi à (XA,YA) de (SEGMENT6, xmin, ymin, xmax, ymax, difx, dify, seuil, erreur) ;
      Envoi à (XB,YB) de (SEGMENT6, xmin, ymin, xmax, ymax, difx, dify, seuil, erreur) ;
   FIN ;

PROCEDURE SEGMENT6_CELLULE (I,J) ;

   DEBUT
      SI NON Marque ALORS
         DEBUT
            Reception (xmin, ymin, xmax, ymax, difx, dify, seuil, erreur) ;
            Aerreur := ABS (erreur) ;
            SI  (Aerreur<=2*seuil)  ALORS
               DEBUT
                  Transmission (1,0) de (SEGMENT6, xmin, ymin, xmax, ymax, difx, dify ,
                                         seuil,  erreur-dify)  ;
                  Transmission (-1,0) de (SEGMENT6, xmin, ymin, xmax, ymax, difx, dify,
                                         seuil,  erreur+dify)  ;
                  Transmission (0,1) de (SEGMENT6, xmin, ymin, xmax, ymax, difx, dify,
                                         seuil,  erreur+difx)  ;
                  Transmission (0,-1) de (SEGMENT6, xmin, ymin, xmax, ymax, difx, dify,

                                         seuil,  erreur-difx)  ;
                  SI  ((xmin<= I)  ET  (I<= xmax)  ET (ymin<=J)  ET  (J<= ymax)
                  ET  (Aerreur<= seuil))  ALORS
                     Afficher;
                  Marquer;
               FIN
         FIN
   FIN ;
```

Remarque :

Pour une propagation limitée classique il faut remplacer le test sur Aerreur par le test de limitation au rectangle englobant.

Dans le cas d'une diffusion physique, il faut comme précédemment incrémenter le paramètre en fonction du port de réception. Il serait facile en outre de supprimer physiquement le "retour à l'envoyeur" pour la diffusion suivante.

ALGORITHME DE TRACE DE SEGMENTS PARALLELE N°7

BRESENHAM A 8 PARAMETRES

```
PROCEDURE SEGMENT7_HOTE ;

   DEBUT
      difx := XB -XA ;
      dify := YB - YA ;
      diax := SGN (difx) ;
      diay := SGN (dify) ;
      max := ABS (difx) ;
      min := ABS (dify) ;
      SI  (max <= min)  ALORS
         DEBUT
            echange := min ;
            min := max ;
            max := echange ;
            axix := 0 ;
            axiy := diay ;
         FIN
      SINON
         DEBUT
            axix := diax ;
            axiy := 0 ;
         FIN ;
      compteur := 0 ;
      seuil := max DIV 2 ;
      Envoi à (XA,YA) de (SEGMENT7, compteur, seuil, min, max, diax, diay, axix, axiy) ;
   FIN ;

PROCEDURE SEGMENT7_CELLULE (I,J) ;

   DEBUT
      Reception (compteur, seuil, min, max, diax, diay, axix, axiy) ;
      SI  (compteur <= max)  ALORS
         DEBUT
            compteur := compteur + 1 ;
            seuil := seuil + min ;
            SI  (seuil >= max)  ALORS
               DEBUT
                  seuil := seuil - max ;
                  Transmission (diax,diay) de (SEGMENT7, compteur, seuil, min, max,
                                               diax, diay, axix, axiy) ;
               FIN
            SINON
               Transmission (axix,axiy) de (SEGMENT7, compteur, seuil, min, max, diax,
                                               diay, axix, axiy) ;
            Afficher;
         FIN
   FIN ;
```

ALGORITHME DE TRACE DE SEGMENTS PARALLELE N°8

BRESENHAM A 5 PARAMETRES

```
PROCEDURE SEGMENT8_HOTE ;

  DEBUT
    SI (XB > XA) ALORS
      DEBUT
        x := XA ;
        y := YA ;
        difx := XB - XA ;
        dify := YB - YA ;
      FIN
    SINON
      DEBUT
        x := XB ;
        y := YB ;
        difx := XA - XB ;
        dify := YA - YB ;
      FIN ;
    SI (dify > 0) ALORS
      SI (difx < dify) ALORS
        indicateur := 1
      SINON
        indicateur := 2
    SINON
      SI (difx > -dify) ALORS
        indicateur := 3
      SINON
        indicateur := 4 ;
    compteur := 0 ;
    erreur := 0 ;
    Envoi à (x,y) de (SEGMENT8, compteur, erreur, difx, dify, indicateur) ;
  FIN ;
```

```
PROCEDURE SEGMENT8_CELLULE (I,J) ;

  DEBUT
    Reception (compteur, erreur, difx, dify, indicateur) ;
    CAS indicateur DE
      1 :
        DEBUT
          compteur := compteur + 1 ;
          SI (compteur <= dify) ALORS
            SI (erreur < 0) ALORS
              DEBUT
                erreur := erreur + 2*dify - 2*difx ;
                Transmission (1,1) de (SEGMENT8, compteur, erreur, difx, dify,
                                       indicateur) ;
              FIN
            SINON
              DEBUT
                erreur := erreur - 2*difx ;
                Transmission (0,1) de (SEGMENT8, compteur, erreur, difx, dify,
                                       indicateur) ;
              FIN
        FIN ;
      2 :
        DEBUT
          compteur := compteur + 1 ;
          SI (compteur <= difx) ALORS
            SI (erreur > 0) ALORS
              DEBUT
                erreur := erreur + 2*dify - 2*difx ;
                Transmission (1,1) de (SEGMENT8, compteur, erreur, difx, dify,
                                       indicateur) ;
              FIN
            SINON
              DEBUT
                erreur := erreur + 2*dify ;
                Transmission (1,0) de (SEGMENT8, compteur, erreur, difx, dify,
                                       indicateur) ;
              FIN
        FIN ;
      3 :
        DEBUT
          compteur := compteur + 1 ;
          SI (compteur <= difx) ALORS
            SI (erreur < 0) ALORS
              DEBUT
                erreur := erreur + 2*dify + 2*difx ;
                Transmission (1,-1) de (SEGMENT8, compteur, erreur, difx, dify,
                                        indicateur) ;
              FIN
            SINON
              DEBUT
                erreur := erreur + 2*dify ;
                Transmission (1,0) de (SEGMENT8, compteur, erreur, difx, dify,
                                       indicateur) ;
              FIN
        FIN ;
```

```
      4 :
         DEBUT
            compteur := compteur + 1 ;
            SI  (compteur <= -dify)  ALORS
               SI  (erreur > 0)  ALORS
                  DEBUT
                     erreur := erreur + 2*dify + 2*difx ;
                     Transmission (1,-1) de (SEGMENT8, compteur, erreur, difx, dify,
                                             indicateur) ;
                  FIN
               SINON
                  DEBUT
                     erreur := erreur + 2*difx ;
                     Transmission (0,-1) de (SEGMENT8, compteur, erreur, difx, dify,
                                             indicateur) ;
                  FIN
         FIN ;
   FIN ;
   Afficher;
FIN ;
```

ALGORITHME DE TRACE DE SEGMENTS PARALLELE N°9

D.D.A.

```
PROCEDURE SEGMENT9_HOTE ;

   DEBUT
      difx := XB - XA ;
      dify := YB - YA ;
      adifx := ABS (difx) ;
      adify := ABS (dify) ;
      SI  (adifx > adify)  ALORS
         DEBUT
            longueur := adifx ;
            indicateur := 0 ;
            SI  (longueur=0)  ALORS
               incrément := 0 ;
            SINON
               increment := dify / longueur ;
            SI  (difx > 0)  ALORS
               Envoi à (XA,YA) de (SEGMENT9,longueur, indicateur, increment, YA)
            SINON
               Envoi à (XB,YB) de (SEGMENT9,longueur, indicateur, -increment, YB)
         FIN
      SINON
         DEBUT
            longueur := adify ;
            indicateur := 1 ;
            SI  (longueur=0)  ALORS
               incrément := 0 ;
            SINON
               increment := difx / longueur ;
            SI  (dify > 0)  ALORS
               Envoi à (XA,YA) de (SEGMENT9,longueur, indicateur, increment, XA)
            SINON
               Envoi à (XB,YB) de (SEGMENT9,longueur, indicateur, -increment, XB)
         FIN
   FIN ;
```

```
PROCEDURE SEGMENT9_CELLULE (I,J) ;

  DEBUT
    Reception (compteur, indicateur, increment, coordonnee) ;
    compteur := compteur - 1 ;
    SI (compteur > 0) ALORS
      DEBUT
        coordonnee := coordonnee + increment ;
        coordsuiv := ARRONDI (coordonnee) ;
        SI (indicateur = 0) ALORS
              Transmission (1,coordsuiv-J) de (SEGMENT9, compteur, indicateur,
                                               increment, coordonnee)
        SINON
              Transmission (coordsuiv-I,1) de (SEGMENT9, compteur, indicateur,

                                               increment, coordonnee) ;
      FIN ;
    Afficher;
  FIN ;
```

ALGORITHME DE TRACE DE SEGMENTS PARALLELE N°10

LOCEFF

```
PROCEDURE SEGMENT10_HOTE ;

  DEBUT
    SI (YA=YB) ALORS
      Envoi à (0,YA) de (SEGMENT10, XA, XB) ;
    SINON
      DEBUT
        difx := XB - XA ;
        SI (difx < 0) ALORS
          DEBUT
            difx := -difx ;
            echange := XA ;
            XA := XB ;
            XB := echange ;
            echange := YA ;
            YA := YB ;
            YB := echange ;
          FIN ;
        dify := YB - YA ;
        xincrement := difx / (ABS (dify) + 1) ;
        yincrement := SGN (dify) ;
        x := XA ;
        y := YA ;
        debut := XA ;
        TANT QUE y <> YB FAIRE
          DEBUT
            x := x + xincrement ;
            fin := ARRONDI (x) ;
            Envoi à (0,y) de (SEGMENT10, debut, fin) ;
            y := y + yincrement ;
            debut := fin ;
          FIN ;
        Envoi à (0,YB) de (SEGMENT10, debut, XB) ;
    FIN ;
  FIN ;


PROCEDURE SEGMENT10_CELLULE (I,J) ;

  DEBUT
    Reception (debut, fin) ;
    Transmission (1,0) de (SEGMENT10, debut, fin) ;
    SI ((debut <= I) ET (I <= fin)) ALORS
      Afficher;
  FIN ;
```

Remarques :

L'algorithme est présenté uniquement pour les segments pour lesquels :

ABS(XA - XB) < ABS(YA-YB)

Pour les autres il suffit d'inverser les rôles respectifs de X et Y. Pour l'évaluation, on ajoutera un test permettant d'aiguiller vers l'un ou l'autre des algorithmes.

ALGORITHME DE TRACE DE SEGMENTS PARALLELE N°11

DICHOTOMIQUE

```
PROCEDURE SEGMENT11_HOTE ;

  DEBUT
    xm := (XA+XB) DIV 2 ;
    ym := (YA+YB) DIV 2 ;
    Envoi à (xm,ym) de (SEGMENT11, XA, YA, XB, YB) ;
  FIN ;


PROCEDURE SEGMENT11_CELLULE (I,J) ;

  DEBUT
    Reception (x1, y1, x2, y2) ;
    xm := ARRONDI ((x1+I) / 2) ;
    ym := ARRONDI ((y1+J) / 2) ;
    SI ((xm <> I) OU (ym <> J)) ALORS
       Transmission_distante (xm, ym) de (SEGMENT11, x1, y1, I, J) ;
    xm := ARRONDI ((x2+I) / 2) ;
    ym := ARRONDI ((y2+J) / 2) ;
    SI ((xm <> I) OU (ym <> J)) ALORS
       Transmission_distante (xm,ym) de (SEGMENT11, I, J, x2, y2) ;
    Afficher;
  FIN ;
```

ANNEXE B

LES ALGORITHMES DE TRACE DE CERCLES

REMARQUES PRELIMINAIRES RELATIVES AUX ALGORITHMES DE TRACES DE CERCLES

Nous présentons en premier lieu les algorithmes séquentiels qui ont servi de base au développement des solutions parallèles proposées à la suite.

L'objet de ces algorithmes est de tracer un cercle de rayon : rayon, et de centre : C(xcentre,ycentre), les coordonnées sont des données globales supposées connues de l'ordinateur hôte (où sont également implémentées les solutions séquentielles).

Les points de l'image et les cellules sont repérés grâce à un même repère orthogonal dont l'origine se situe dans le coin inférieur droit (de l'image où du réseau).

Chaque cellule est supposée connaître ses propres coordonnées (I,J), ce qui n'est pas indispensable pour toutes les solutions présentées.

La procédure CERCLEX (I,J) correspond au programme exécuté une fois que la cellule a analysé la première partie du message reçu, à savoir le nom de la commande, en l'occurence CERCLEX.

Rappel des notations utilisées :

- Envoi à (x,y) de (commande paramétrée) : Emission d'une commande depuis l'ordinateur hôte à destination de la cellule (x,y).

- Reception (paramètres de commandes) : Reception par la cellule des paramètres associés à la commande qu'elle vient d'interpréter.

- Transmission (i,j) de (commande paramétrée) : Transmission d'une commande, d'une cellule vers une de ses voisines directes (i et j prennent leur valeur parmi -1,0,1).

  Remarque :
  Pour les solutions multipipelines, nous conservons cette même notation par souci d'homogénéité, mais il est évident que les paramètres i et j sont superflus, puisque toutes les transmissions sont identiques : Transmission (1,0) de (commande) par exemple.

- Afficher sans paramètre : est utilisée dans les procédures exécutées par les cellules pour allumer le pixel qui leur est associé.

- AfficherX (x,y) : sont utilisées dans les solutions séquentielles pour afficher les pixels associés à (x,y) par le truchement des symétries, elles seront détaillées pour chacune des solutions.

Remarque importante :
Les boucles POUR présentes dans les procédures de l'ordinateur hôte ne sont là que pour faciliter la compréhension. Il ne faut pas prendre en compte le caractère séquentiel qu'elles renferment. Elles correspondent dans la pratique à une diffusion *'quasiment simultanée'* d'une commande à tout ou partie du réseau. Il n'est pas du ressort de cette thèse d'étudier la manière dont l'ordinateur hôte transmet physiquement ses commandes au réseau.

Fonctions mathématiques :

- INF (x,y) : retourne la plus petite des deux valeurs x et y.

- SUP (x,y) : retourne la plus grande des deux valeurs x et y.

- ABS (x) : retourne la valeur absolue de x.

- SGN (x) : retourne 1 si l'argument est positif, 0 si l'argument est nul, -1 sinon.

- ARRONDI (x) : retourne l'entier le plus proche du réel x.

- DIV : opérateur de division entière. Rappelons d'ailleurs que (DIV 2) et (* 2) correspondent à de simples décalages et ne posent pas de problèmes pour le câblage éventuel.

- MAXINT : entier le plus grand.

ALGORITHME DE TRACE DE CERCLES SEQUENTIEL N°1

BRESENHAM OCTAL

```
PROCEDURE CERCLE1 ;

  DEBUT
    x := rayon ;
    y := 0 ;
    erreury := 0 ;
    TANT QUE x>y FAIRE
      DEBUT
        Affichage8 (x,y) ;
        erreury := erreury + 2*y + 1 ;
        erreurxy := erreury - 2*x + 1 ;
        SI (ABS (erreury) >= ABS (erreurxy)) ALORS
          DEBUT
            x := x - 1 ;
            erreury := erreurxy ;
          FIN
        y := y + 1 ;
      FIN ;
    SI (x=y) ALORS
      Affichage8 (x,y) ;
  FIN ;
```

Remarque :

Affichage8 (x,y) : Affiche les huit points associés à (x,y) en appliquant les symétries par rapport au centre du cercle.

ALGORITHME DE TRACE DE CERCLES SEQUENTIEL N°2

BRESENHAM QUATERNAIRE

```
PROCEDURE CERCLE2 ;

  DEBUT
    x := rayon ;
    y := 0 ;
    erreur := 0 ;
    TANT QUE x>0 FAIRE
      DEBUT
        Affichage4 (x,y) ;
        erreurx := erreur - 2*x + 1 ;
        erreury := erreur + 2*y + 1 ;
        SI (ABS (erreurx) >= ABS (erreury)) ALORS
          DEBUT
            y := y + 1 ;
            erreur := erreury ;
          FIN
        SINON
          DEBUT
            x := x - 1 ;
            erreur := erreurx ;
          FIN
      FIN ;
    SI (x=0) ALORS
      Affichage4 (x,y) ;
  FIN ;
```

Remarque :

Afficher4 (x,y) : Affiche les quatre points associés à (x,y), en appliquant les symétries par rapport au centre du cercle.

ALGORITHME DE TRACE DE CERCLES SEQUENTIEL N°3

BISWAS

```
PROCEDURE CERCLE3 ;

  DEBUT
    lgsup := (2 * rayon + 1)2 ;
    lginf := (2 * rayon - 1)2 ;
    erreur := 4 * rayon2 ;
    x := rayon ;
    y := 0 ;
    SI (x <> 0) ALORS
       SI (erreur <= lgsup) ALORS
          DEBUT
             SI (erreur >= lginf) ALORS
                Afficher1 (x,y) ;
             y := y + 1 ;
             erreur := erreur + 8*y + 4 ;
          FIN
       SINON
          DEBUT
             x := x - 1 ;
             y := y - 1 ;
             erreur := erreur + 8*y + 4 ;
          FIN
  FIN ;
```

Remarque :

Afficher1 (x,y) : Affiche le pixel (x,y).

ALGORITHME DE TRACE DE CERCLES SEQUENTIEL N°4

D.D.A.

```
PROCEDURE CERCLE4 ;
  DEBUT
     sinus := 1 / sqrt (rayon2 + 1) ;
     cosinus := rayon * sinus ;
     x := rayon ;
     y := 0 ;
     memx := 0 ;
     TANT QUE (ARRONDI (memx) <> rayon) ET (ARRONDI (y) <> 0) FAIRE
        DEBUT
           Afficher1 (ARRONDI (x), ARRONDI (y)) ;
           memx := x*cosinus + y*sinus ;
           y := y*cosinus - x*sinus ;
           x := memx ;
        FIN
  FIN ;
```

Remarque :

Afficher1 (x,y) : Affiche le pixel (x,y).

ALGORITHME DE TRACE DE CERCLES SEQUENTIEL N°5

MORVAN

```
PROCEDURE CERCLE5 ;

  DEBUT
    x := rayon ;
    y := 0 ;
    xarrondi := 0 ;
    yarrondi := 0 ;
    TANT QUE (xarrondi<>rayon) ET (yarrondi<>rayon) FAIRE
      DEBUT
        x := x - y/rayon ;
        y := y + x/rayon ;
        xarrondi := ARRONDI (x) ;
        yarrondi := ARRONDI (y) ;
        Afficher1 (xarrondi,yarrondi) ;
      FIN
  FIN ;
```

Remarque :

Afficher1 (x,y) : Affiche le pixel (x,y).

ALGORITHME DE TRACE DE CERCLES PARALLELE N°1

S.I.M.D.

```
PROCEDURE CERCLE1_HOTE ;

  DEBUT
    raycar := rayon^2
    POUR I := 0 A Taille_reseau FAIRE
      POUR j := 0 A Taille_reseau FAIRE
        Envoi à (i,j) de (CERCLE1, raycar, xcentre, ycentre) ;
  FIN ;

PROCEDURE CERCLE1_CELLULE (I,J) ;

  DEBUT
    Reception (raycar, xcentre, ycentre) ;
    xrelatif := I - xcentre ;
    yrelatif := J - ycentre ;
    erreur := ABS (raycar - xrelatif^2 - yrelatif^2 ) ;
    seuil := 2 * SUP (ABS (xrelatif), ABS (yrelatif)) + 1 ;
    SI (2*erreur <= seuil) ALORS
      Afficher ;
  FIN ;
```

ALGORITHME DE TRACE DE CERCLES PARALLELE N°2

MULTIPIPELINE

```
PROCEDURE CERCLE2_HOTE ;

   DEBUT
      erreur := -rayon2 + xcentre2 ;
      xrelatif := -xcentre ;
      POUR j:= ycentre-rayon A ycentre+rayon FAIRE
         Envoi à (0,j) de (CERCLE2, erreur, xrelatif, ycentre) ;
   FIN ;


PROCEDURE CERCLE2_CELLULE (I,J) ;

   DEBUT
      Reception (erreur, xrelatif, ycentre) ;
      Transmission (1,0) de (CERCLE2, erreur+2*xrelatif+1, xrelatif+1, ycentre) ;
      yrelatif := J - ycentre ;
      erreur := 2 * ABS (erreur + yrelatif2) ;
      seuil := 2 * SUP (ABS (xrelatif), ABS (yrelatif)) + 1 ;
      SI (erreur <= seuil) ALORS
         Afficher ;
   FIN ;
```

ALGORITHME DE TRACE DE CERCLES PARALLELE N°3

MULTIPIPELINE A VECTEUR

```
PROCEDURE CERCLE3_HOTE ;

   DEBUT
      erreur := 2 * (xcentre2 + 2*rayon + 1) ;
      POUR j := 0 A 2*rayon FAIRE
         DEBUT
            D := j - rayon ;
            Ayrelatif := ABS (D) ;
            erreur := erreur + 4*D - 2 ;
            xrelatif := -xcentre ;
            Envoi à (0,ycentre + D) de (CERCLE3, Ayrelatif, erreur, xrelatif);
         FIN ;
   FIN ;


PROCEDURE CERCLE3_CELLULE (I,J) ;

   DEBUT
      Reception (erreur, xrelatif, Ayrelatif) ;
      Transmission (1,0) de (erreur+4*xrelatif+2, xrelatif+1, Ayrelatif) ;
      seuil := 2 * SUP (ABS (xrelatif), Ayrelatif) + 1 ;
      SI (ABS(erreur) <= seuil) ALORS
         Afficher ;
   FIN ;
```

Remarque :

Pour les algorithmes à propagation qui vont suivre, l'incrémentation des paramètres est faite avant transmission. Une autre solution consiste à l'effectuer lors de la réception en fonction du port de communication sur lequel la commande est arrivée. Les tests sont alors remplacés par un "case" sur le port de reception, ce qui est un peu meilleur pour l'évaluation des performances.
Il s'agit seulement d'une astuce technique qui n'influe pas sur le comportement global des algorithmes.

ALGORITHME DE TRACE DE CERCLES PARALLELE N°4

PROPAGATION PURE

```
PROCEDURE CERCLE4_HOTE ;

  DEBUT
    erreur = xcentre2 + ycentre2 - rayon2 ;
    xrelatif = - xcentre ;
    yrelatif = - ycentre ;
    Envoi à (xcentre, ycentre) de (CERCLE4, erreur, xrelatif, yrelatif) ;
  FIN ;


PROCEDURE CERCLE4_CELLULE (I,J) ;

  DEBUT
    SI (NON Marque) ALORS
      DEBUT
        Reception (erreur, xrelatif, yrelatif) ;
        POUR deplac1 := -1 A 1 FAIRE
          POUR deplac2 := -1 A 1 FAIRE
            DEBUT
              test := faux ;
              errtrans := erreur ;
              xreltrans := xrelatif ;
              yreltrans := yrelatif ;
              SI (deplac1 <> 0) ALORS
                DEBUT
                  test := vrai ;
                  errtrans := erreur + 2 * deplac1 * xrelatif + 1 ;
                  xreltrans := xrelatif + deplac1 ;
                FIN ;
              SI (deplac2 <> 0) ALORS
                DEBUT
                  test := vrai ;
                  errtrans := erreur + 2 * deplac2 * yrelatif + 1 ;
                  yreltrans := yrelatif + deplac2 ;
                FIN ;
              SI (test) ALORS
                Transmission (deplac1,deplac2) de (CERCLE4, errtrans, xreltrans,
                                                   yreltrans) ;
            FIN ;
        seuil := 2 * SUP (ABS (xrelatif), ABS (yrelatif)) + 1 ;
        Marquer ;
        SI (2 * ABS (erreur) <= seuil) ALORS
          Afficher ;
      FIN ;
  FIN ;
```

ALGORITHME DE TRACE DE CERCLES PARALLELE N°5

PROPAGATION LIMITEE

```
PROCEDURE CERCLE5_HOTE ;

  DEBUT
    erreur = -rayon2 ;
    xrelatif = 0 ;
    yrelatif = 0 ;
    Envoi à (xcentre, ycentre) de (CERCLE5, erreur, xrelatif, yrelatif, rayon) ;
  FIN ;

PROCEDURE CERCLE5_CELLULE (I,J) ;

  DEBUT
    Reception (erreur, xrelatif, yrelatif, rayon) ;
    SI  ((ABS(xrelatif)<=rayon)  ET  (ABS(yrelatif)<=rayon))  ALORS
      DEBUT
        SI (NON Marque) ALORS
        DEBUT
          POUR deplac1 := -1  A  1  FAIRE
            POUR deplac2 := -1  A  1  FAIRE
              DEBUT
                test := faux ;
                errtrans := erreur ;
                xreltrans := xrelatif ;
                yreltrans := yrelatif ;
                SI (deplac1 <> 0) ALORS
                  DEBUT
                    test := vrai ;
                    errtrans := erreur + 2 * deplac1 * xrelatif + 1 ;
                    xreltrans := xrelatif + deplac1 ;
                  FIN ;
                SI (deplac2 <> 0) ALORS
                  DEBUT
                    test := vrai ;
                    errtrans := erreur + 2 * deplac2 * yrelatif + 1 ;
                    yreltrans := yrelatif + deplac2 ;
                  FIN ;
                SI (test) ALORS
                  Transmission (deplac1,deplac2) de (CERCLE5, errtrans,
                                                     xreltrans, yreltrans, rayon) ;
              FIN ;
          seuil := 2 * SUP (ABS (xrelatif), ABS (yrelatif)) + 1 ;
          Marquer ;
          SI  (2 * ABS (erreur) <= seuil)  ALORS
            Afficher ;
        FIN ;
    FIN ;
  FIN ;
```

ALGORITHME DE TRACE DE CERCLES PARALLELE N°6

PROPAGATION INTELLIGENTE

```
PROCEDURE CERCLE6_HOTE ;

  DEBUT
    erreur = -rayon2 ;
    xrelatif = 0 ;
    yrelatif = 0 ;
    Envoi à (xcentre, ycentre) de (CERCLE6, erreur, xrelatif, yrelatif) ;
  FIN ;

PROCEDURE CERCLE6_CELLULE (I,J) ;

  DEBUT
    Reception (erreur, xrelatif, yrelatif) ;
    SI  ((ABS(xrelatif)<=rayon)  ET  (ABS(yrelatif)<=rayon))  ALORS
      DEBUT
      SI  (xrelatif >= 0)  ALORS
        DEBUT
          SI  (yrelatif >= 0)  ALORS
            Transmission (1,0) de (CERCLE6, erreur + 2 * xrelatif + 1,
                xrelatif + 1, yrelatif)) ;
          SI  (yrelatif <= 0)  ALORS
            Transmission (0,-1) de (CERCLE6, erreur - 2 * yrelatif + 1,
                xrelatif, yrelatif - 1)) ;
        FIN ;
      SI  (xrelatif <= 0)  ALORS
        DEBUT
          SI  (yrelatif >= 0)  ALORS
            Transmission (0,1) de (CERCLE6, erreur + 2 * yrelatif + 1,
                xrelatif, yrelatif + 1)) ;
          SI  (yrelatif <= 0)  ALORS
            Transmission (-1,0) de (CERCLE6, erreur - 2 * xrelatif + 1,
                xrelatif - 1, yrelatif)) ;
        FIN ;
      seuil := 2 * SUP (ABS (xrelatif), ABS (yrelatif)) + 1 ;
      SI  (2 * ABS (erreur) <= seuil)  ALORS
        Afficher
      FIN ;
  FIN ;
```

ALGORITHME DE TRACE DE CERCLES PARALLELE N°7

PROPAGATION INTELLIGENTE AMELIOREE

```
PROCEDURE CERCLE7_HOTE ;

  DEBUT
    erreur = -rayon2 ;
    xrelatif = 0 ;
    yrelatif = 0 ;
    Envoi à (xcentre, ycentre) de (CERCLE7, erreur, xrelatif, yrelatif) ;
  FIN ;
```

```
PROCEDURE CERCLE7_CELLULE (I,J) ;

  DEBUT
    Reception (erreur, xrelatif, yrelatif) ;
    SI  ((ABS(xrelatif)<=rayon)  ET  (ABS(yrelatif)<=rayon))  ALORS
      DEBUT
        deplac1 := SGN (xrelatif) ;
        deplac2 := SGN (yrelatif) ;
        SI (deplac2 = 0) ALORS
          SI (deplac1 = 0) ALORS
            DEBUT
              Transmission (1,0) de (CERCLE7, erreur + 2 * xrelatif + 1,
                xrelatif + 1, yrelatif));
              Transmission (-1,0) de (CERCLE7, erreur - 2 * xrelatif + 1,
                xrelatif - 1, yrelatif));
              Transmission (0,1) de (CERCLE7, erreur + 2 * yrelatif + 1,
                xrelatif, yrelatif + 1));
              Transmission (0,-1) de (CERCLE7, erreur - 2 * yrelatif + 1,
                xrelatif, yrelatif - 1));
            FIN
          SINON
            DEBUT
              Transmission (deplac1,0) de (CERCLE7,
                erreur + 2 * xrelatif * deplac1 + 1, xrelatif + deplac1, yrelatif)) ;
              Transmission (0,-deplac1) de (CERCLE7,
                erreur - 2 * yrelatif * deplac1 + 1, xrelatif, yrelatif - deplac1)) ;
            FIN
        SINON
          SI (deplac1 = 0) ALORS
            DEBUT
              Transmission (deplac2,0) de (CERCLE7,
                erreur + 2 * deplac2 * xrelatif + 1, xrelatif + deplac2, yrelatif)) ;
              Transmission (0,deplac2) de (CERCLE7,
                erreur + 2 * deplac2 * yrelatif + 1, xrelatif, yrelatif + deplac2)) ;
            FIN
          SINON
            DEBUT
              deplac1 := SGN (deplac1 + deplac2) ;
              deplac2 := SGN (deplac2 - deplac1) ;
              Transmission (deplac1,deplac2) de (CERCLE7,
                erreur + 2 * deplac1 * xrelatif+ 2 * deplac2 * yrelatif + 1,
                xrelatif + deplac1, yrelatif + deplac2)) ;
            FIN ;
        seuil := 2 * SUP (ABS (xrelatif), ABS (yrelatif)) + 1 ;
        SI  (2 * ABS (erreur) <= seuil)  ALORS
          Afficher ;
      FIN ;
  FIN ;
```

ALGORITHME DE TRACE DE CERCLES PARALLELE N°8

PROPAGATION OCTAL

```
PROCEDURE CERCLE8_HOTE ;

  DEBUT
    indicateur = 16 ;
    erreur = -rayon2 ;
    xrelatif = 0 ;
    yrelatif = 0 ;
    Envoi à (xcentre, ycentre) de (CERCLE8, indicateur, erreur, xrelatif, yrelatif) ;
  FIN ;
```

```
PROCEDURE CERCLE8_CELLULE (I,J) ;

  DEBUT
    Reception (indicateur, erreur, xrelatif, yrelatif) ;
    SI  ((ABS(xrelatif)<=rayon)  ET  (ABS(yrelatif)<=rayon))  ALORS
      DEBUT
        SI (indicateur > 7) ALORS
          direction := indicateur - 8
        SINON
          direction := indicateur ;
        CAS direction DE
          0 :
            DEBUT
              Transmission (1,0) de (CERCLE8, direction, erreur + 2 * xrelatif + 1,
                                                    xrelatif + 1, yrelatif)) ;
              SI (indicateur > 7) ALORS
                Transmission (1,-1) de (CERCLE8, indicateur,
                erreur + 2 * (xrelatif - yrelatif + 1), xrelatif + 1, yrelatif - 1));
            FIN ;
          1 :
            DEBUT
              Transmission (1,1) de (CERCLE8, direction,
                erreur + 2 * (xrelatif + yrelatif + 1), xrelatif + 1, yrelatif + 1));
              SI (indicateur > 7) ALORS
                Transmission (1,0) de (CERCLE8, indicateur,
                  erreur + 2 * xrelatif + 1,  xrelatif + 1, yrelatif)) ;
            FIN ;
          2 :
            DEBUT
              Transmission (0,1) de (CERCLE8, direction, erreur + 2 * yrelatif + 1,
                                                    xrelatif, yrelatif + 1)) ;
              SI (indicateur > 7) ALORS
                Transmission (1,1) de (CERCLE8, indicateur,
                erreur + 2 * (xrelatif + yrelatif + 1), xrelatif + 1, yrelatif + 1));
            FIN ;
          3 :
            DEBUT
              Transmission (-1,1) de (CERCLE8, direction,
                erreur + 2 * (yrelatif - xrelatif + 1), xrelatif - 1, yrelatif + 1)) ;
              SI (indicateur > 7) ALORS
                Transmission (0,1) de (CERCLE8, indicateur,
                  erreur + 2 * yrelatif + 1, xrelatif,  yrelatif + 1)) ;
            FIN ;
          4 :
            DEBUT
              Transmission (-1,0) de (CERCLE8, direction,
                erreur - 2 * xrelatif + 1, xrelatif - 1, yrelatif)) ;
              SI (indicateur > 7) ALORS
                Transmission (-1,1) de (CERCLE8, indicateur,
              erreur + 2 * (-xrelatif + yrelatif + 1),  xrelatif - 1, yrelatif + 1));
            FIN ;
```

```
        5 :
            DEBUT
               Transmission (-1,-1) de (CERCLE8, direction,
                erreur + 2 * (-xrelatif - yrelatif + 1), xrelatif - 1, yrelatif - 1));
               SI (indicateur > 7) ALORS
                  Transmission (-1,0) de (CERCLE8, indicateur,
                     erreur - 2 * xrelatif + 1,  xrelatif - 1, yrelatif)) ;
            FIN ;
        6 :
            DEBUT
               Transmission (0,-1) de (CERCLE8, direction,
                  erreur - 2 * yrelatif + 1, xrelatif, yrelatif - 1)) ;
               SI (indicateur > 7) ALORS
                  Transmission (-1,-1) de (CERCLE8, indicateur,
                  erreur + 2 * (-xrelatif - yrelatif + 1),  xrelatif - 1, yrelatif - 1));
            FIN ;
        7 :
            DEBUT
               Transmission (1,-1) de (CERCLE8, direction,
                  erreur + 2 * (xrelatif - yrelatif + 1), xrelatif + 1, yrelatif - 1)) ;
               SI (indicateur > 7) ALORS
                  Transmission (0,-1) de (CERCLE8, indicateur,
                     erreur - 2 * yrelatif + 1,  xrelatif, yrelatif - 1)) ;
            FIN ;
        8 :
            DEBUT
               Transmission (1,-1) de (CERCLE8, 15,
                  erreur + 2 * (xrelatif - yrelatif + 1), xrelatif + 1, yrelatif - 1)) ;
               Transmission (0,-1) de (CERCLE8, 14,
                  erreur - 2 * yrelatif + 1, xrelatif, yrelatif - 1)) ;
               Transmission (-1,-1) de (CERCLE8, 13,
                  erreur + 2 * (-xrelatif - yrelatif + 1),  xrelatif - 1, yrelatif - 1));
               Transmission (-1,0) de (CERCLE8, 12,
                  erreur - 2 * xrelatif + 1, xrelatif - 1, yrelatif)) ;
               Transmission (-1,1) de (CERCLE8, 11,
                  erreur + 2 * (yrelatif - xrelatif + 1), xrelatif - 1, yrelatif + 1)) ;
               Transmission (0,1) de (CERCLE8, 10, erreur + 2 * yrelatif + 1,
                  xrelatif, yrelatif + 1)) ;
               Transmission (1,1) de (CERCLE8, 9,
                  erreur + 2 * (xrelatif + yrelatif + 1),  xrelatif + 1, yrelatif + 1)) ;
               Transmission (1,0) de (CERCLE8, 8, erreur + 2 * xrelatif + 1,
                  xrelatif + 1, yrelatif)) ;
            FIN
      FIN ;
      seuil := 2 * SUP (ABS (xrelatif), ABS (yrelatif)) + 1 ;
      SI  (2 * ABS (erreur) <= seuil)  ALORS
         Afficher ;
   FIN ;
FIN ;
```

Remarque :

Ce n'est certes pas l'écriture du programme la plus élégante ni la plus concise, mais elle est performante. Une fois micro programmé et en adoptant un rangement adéquat, l'aiguillage est rapide.

ALGORITHME DE TRACE DE CERCLES PARALLELE N°9

BRESENHAM A 7 POINTS

```
PROCEDURE CERCLE9_HOTE ;

   DEBUT
      approx := ARRONDI (rayon * 0.707) ;
      erreur := -rayon2 + 2 * approx2 ;
      Envoi à (xcentre + approx, ycentre + approx) de (CERCLE9, 1, erreur, approx,
         approx) ;
      Envoi à (xcentre + approx, ycentre - approx) de (CERCLE9, 8, erreur, approx,
         -approx) ;
      Envoi à (xcentre, ycentre + rayon) de (CERCLE9, 2, 0, 0, rayon) ;
      Envoi à (xcentre, ycentre - rayon) de (CERCLE9, 7, 0, 0, -rayon) ;
      Envoi à (xcentre - approx, ycentre + approx) de (CERCLE9, 3, erreur,
         -approx, approx) ;
      Envoi à (xcentre - approx, ycentre - approx) de (CERCLE9, 6, erreur, -approx,
         -approx) ;
      Envoi à (xcentre - rayon, ycentre) de (CERCLE9, 4, 0, -rayon, 0) ;
      Envoi à (xcentre - rayon, ycentre) de (CERCLE9, 5, 0, -rayon, 0) ;
   FIN ;


PROCEDURE CERCLE9_CELLULE (I,J) ;

   DEBUT
      Reception (indicateur, erreur, xrelatif, yrelatif) ;
      CAS indicateur DE
         1 :
            DEBUT
               a := 0 ;
               b := yrelatif ;
               i1 := -yrelatif ;
               i2 := xrelatif ;
               deplac1 := 0 ;
               deplac2 := -1 ;
               deplac3 := -1 ;
            FIN ;
         2 :
            DEBUT
               a := xrelatif ;
               b := yrelatif ;
               i1 := xrelatif ;
               i2 := -yrelatif ;
               deplac1 := 1 ;
               deplac2 := 0 ;
               deplac3 := -1 ;
            FIN ;
```

```
3 :
   DEBUT
      a := xrelatif ;
      b := 0 ;
      i1 := xrelatif ;
      i2 := yrelatif ;
      deplac1 := 1 ;
      deplac2 := 0 ;
      deplac3 := 1 ;
   FIN ;
4 :
   DEBUT
      a := xrelatif ;
      b := -yrelatif ;
      i1 := yrelatif ;
      i2 := xrelatif ;
      deplac1 := 0 ;
      deplac2 := 1 ;
      deplac3 := 1 ;
   FIN ;
5 :
   DEBUT
      a := xrelatif ;
      b := yrelatif ;
      i1 := -yrelatif ;
      i2 := xrelatif ;
      deplac1 := 0 ;
      deplac2 := -1 ;
      deplac3 := -1 ;
   FIN ;
6 :
   DEBUT
      a := xrelatif ;
      b := 0 ;
      i1 := xrelatif ;
      i2 := -yrelatif ;
      deplac1 := 1 ;
      deplac2 := 0 ;
      deplac3 := -1 ;
   FIN ;
7 :
   DEBUT
      a := xrelatif ;
      b := -yrelatif ;
      i1 := xrelatif ;
      i2 := yrelatif ;
      deplac1 := 1 ;
      deplac2 := 0 ;
      deplac3 := 1 ;
   FIN ;
8 :
   DEBUT
      a := yrelatif ;
      b := 0 ;
      i1 := yrelatif ;
      i2 := xrelatif ;
      deplac1 := 0 ;
      deplac2 := 1 ;
      deplac3 := 1 ;
   FIN
FIN ;
SI (a <= b) ALORS
   DEBUT
      erreur1 := erreur + 2 * i1 + 1 ;
      erreur2 := erreur1 + 2 * i2 + 1 ;
      SI (ABS (erreur1) <= ABS (erreur2)) ALORS
         Transmission (deplac1,deplac2) de (CERCLE9, indicateur, erreur1,
            xrelatif + deplac1, yrelatif + deplac2))
      SINON
         Transmission (1,deplac3) de (CERCLE9, indicateur, erreur2, xrelatif + 1,
            yrelatif + deplac3)) ;
      Afficher ;
   FIN ;
FIN ;
```

Remarque :

Nous avons approché la racine carrée de 2 par 1,414. Cette approximation améliorant quelque peu les performances ne porte pas préjudice à l'algorithme.

ALGORITHME DE TRACE DE CERCLES PARALLELE N°10

BRESENHAM A 4 POINTS

```
PROCEDURE CERCLE10_HOTE ;

  DEBUT
    Envoi à (xcentre + rayon, ycentre) de (CERCLE10, 1, 0, rayon, 0) ;
    Envoi à (xcentre + rayon, ycentre - 1) de (CERCLE10, 1, 1, rayon, -1) ;
    Envoi à (xcentre, ycentre + rayon) de (CERCLE10, 2, 0, 0, rayon) ;
    Envoi à (xcentre - 1, ycentre + rayon) de (CERCLE10, 2, 1, -1, rayon) ;
    Envoi à (xcentre, ycentre - rayon) de (CERCLE10, 2, 0, 0, -rayon) ;
    Envoi à (xcentre - 1, ycentre - rayon) de (CERCLE10, 2, 1, -1, -rayon) ;
    Envoi à (xcentre - rayon, ycentre) de (CERCLE10, 1, 0, -rayon, 0) ;
    Envoi à (xcentre - rayon, ycentre - 1) de (CERCLE10, 1, 1, -rayon, -1) ;
  FIN ;

PROCEDURE CERCLE10_CELLULE (I,J) ;

  DEBUT
    Reception (indicateur, erreur, xrelatif, yrelatif) ;
    SI (indicateur = 1) ALORS
      DEBUT
        a := ABS (xrelatif) ;
        b := ABS (yrelatif) ;
        deplac1 := 0 ;
        SI (yrelatif < 0) ALORS deplac2 := -1 SINON deplac2 := 1 ;
        deplac3 := -SGN (xrelatif) ;
        deplac4 := deplac2 ;
      FIN
    SINON
      DEBUT
        a := ABS (yrelatif) ;
        b := ABS (xrelatif) ;
        SI (xrelatif < 0) ALORS deplac1 := -1 SINON deplac1 := 1 ;
        deplac2 := 0 ;
        deplac3 := deplac1 ;
        deplac4 := -SGN (yrelatif) ;
      FIN ;
    SI (a > b) ALORS
      DEBUT
        erreur1 := erreur + 2 * b + 1 ;
        erreur2 := erreur1 - 2 * a + 1 ;
        SI (ABS (erreur1) <= ABS (erreur2)) ALORS
          Transmission (deplac1,deplac2) de (CERCLE10, indicateur, erreur1,
                                             xrelatif + deplac1, yrelatif + deplac2))
        SINON
          Transmission (deplac3,deplac4) de (CERCLE10, indicateur, erreur2,
                                             xrelatif + deplac3, yrelatif + deplac4));
      FIN ;
    Afficher ;
  FIN ;
```

ALGORITHME DE TRACE DE CERCLES PARALLELE N°11

BRESENHAM A 3 POINTS

```
PROCEDURE CERCLE11_HOTE ;

  DEBUT
    Envoi à (xcentre, ycentre + rayon) de (CERCLE11, 1, 0, 0, rayon) ;
    Envoi à (xcentre, ycentre - rayon) de (CERCLE11, 1, 0, 0, -rayon) ;
    Envoi à (xcentre - rayon, ycentre) de (CERCLE11, 2, 0, -rayon, 0) ;
    Envoi à (xcentre - rayon, ycentre - 1) de (CERCLE11, 2, 1, -rayon, -1) ;
  FIN ;

PROCEDURE CERCLE11_CELLULE (I,J) ;

  DEBUT
    Reception (indicateur, erreur, xrelatif, yrelatif) ;
    SI (indicateur = 2) ALORS
      DEBUT
        SI (yrelatif < 0) ALORS deplac := -1 SINON deplac := 1 ;
        test := -xrelatif ;
      FIN
    SINON
      DEBUT
        SI (yrelatif < 0) ALORS deplac := 1 SINON deplac := -1 ;
          test := ABS (yrelatif) ;
      FIN ;
    SI (test > 0) ALORS
      DEBUT
        erreurx := erreur + 2 * xrelatif + 1 ;
        erreury := erreur + 2 * deplac * yrelatif + 1 ;
        erreurxy := erreurx + erreury - erreur ;
        Aerreurxy := ABS (erreurxy) ;
        SI (Aerreurxy <= ABS (erreury)) ALORS
          SI (Aerreurxy <= ABS (erreurx)) ALORS
            Transmission (1,deplac) de (CERCLE11, indicateur, erreurxy,
              xrelatif + 1, yrelatif + deplac))
          SINON
            Transmission (1,0) de (CERCLE11, indicateur, erreurx, xrelatif + 1,
              yrelatif))
        SINON
          Transmission (0,deplac) de (CERCLE11, indicateur, erreury, xrelatif,
            yrelatif + deplac))
      FIN ;
    Afficher ;
  FIN ;
```

ALGORITHME DE TRACE DE CERCLES PARALLELE N°12

BRESENHAM A 2 POINTS

```
PROCEDURE CERCLE12_HOTE ;

  DEBUT
    erreur := 0 ;
    xrelatif := 0 ;
    yrelatif := rayon ;
    Envoi à (xcentre, ycentre + rayon) de (CERCLE12,erreur,xrelatif,yrelatif) ;
    yrelatif := -rayon ;
    Envoi à (xcentre, ycentre - rayon) de (CERCLE12,erreur,xrelatif,yrelatif) ;
  FIN ;

PROCEDURE CERCLE12_CELLULE (I,J) ;

  DEBUT
    Reception (erreur, xrelatif, yrelatif) ;
    SI (yrelatif <> 0) ALORS
      DEBUT
        ax := ABS (xrelatif) ;
        ay := ABS (yrelatif) ;
        SI (xrelatif < 0) ALORS deplac1 := -1 SINON deplac1 := 1 ;
        deplac2 := SGN (yrelatif) ;
        erreurx := erreur + 2 * ax + 1 ;
        erreury := erreur - 2 * ay + 1 ;
        erreurxy := erreurx + erreury - erreur ;
        Aerreurxy := ABS (erreurxy) ;
        SI (Aerreurxy <= ABS (erreury)) ALORS
          SI (Aerreurxy <= ABS (erreurx)) ALORS
            Transmission (deplac1,deplac2) de (CERCLE12, erreurxy,
              xrelatif + deplac1, yrelatif - deplac2))
          SINON
            DEBUT
              Transmission (deplac1,0) de (CERCLE12, erreurx, xrelatif + deplac1,
                yrelatif)) ;
              SI (xrelatif = 0) ALORS
                Transmission (-1,0) de (CERCLE12, erreurx, -1, yrelatif)) ;
            FIN
        SINON
          Transmission (0,-deplac2) de (CERCLE12, erreury, xrelatif,
            yrelatif - deplac2)) ;
      FIN ;
    Afficher ;
  FIN ;
```

ALGORITHME DE TRACE DE CERCLES PARALLELE N°13

BRESENHAM A 1 POINT

```
PROCEDURE CERCLE13_HOTE ;

  DEBUT
    erreur := 0 ;
    xrelatif := -rayon ;
    yrelatif := MAXINT ;
    Envoi à (xcentre - rayon, ycentre) de (CERCLE13, erreur, xrelatif, yrelatif) ;
  FIN ;

PROCEDURE CERCLE13_CELLULE (I,J) ;

  DEBUT
    Reception (erreur, xrelatif, yrelatif) ;
    SI (yrelatif = MAXINT) ALORS
      DEBUT
        Transmission (0,1) de (CERCLE13, 1, xrelatif, 1)) ;
        Transmission (0,-1) de (CERCLE13, 1, xrelatif, -1)) ;
      FIN
    SINON
      SI (yrelatif <> 0) ALORS
        DEBUT
          SI (xrelatif < 0) ALORS signex := -1 SINON signex := 1 ;
          SI (yrelatif < 0) ALORS signey := -1 SINON signey := 1 ;
          deplac := signex * signey ;
          erreurx := erreur + 2 * xrelatif + 1 ;
          erreury := erreur - 2 * yrelatif * deplac + 1 ;
          erreurxy := erreurx + erreury - erreur ;
          Aerreurxy := ABS (erreurxy) ;
          SI (Aerreurxy <= ABS (erreury)) ALORS
            SI (Aerreurxy <= ABS (erreurx)) ALORS
              Transmission (1,-deplac) de (CERCLE13, erreurxy, xrelatif + 1,
                                           yrelatif - deplac))
            SINON
              Transmission (1,0) de (CERCLE13, erreurx, xrelatif + 1, yrelatif))
          SINON
            Transmission (0,-deplac) de (CERCLE13, erreury, xrelatif,
              yrelatif - deplac))
        FIN ;
    Afficher ;
  FIN ;
```

Remarque :

Nous utilisons une valeur fictive (MAXINT) pour traiter le cas de la cellule initiale.

ALGORITHME DE TRACE DE CERCLES PARALLELE N°14

BISWAS A 3 POINTS

```
PROCEDURE CERCLE14_HOTE ;

  DEBUT
    lgsup := (2 * rayon + 1)2 ;
    lginf := (2 * rayon - 1)2 ;
    erreur := 4 * rayon2 ;
    Envoi à (xcentre, ycentre + rayon) de (CERCLE14, 1, lgsup, lginf, erreur, 0, rayon);
    Envoi à (xcentre, ycentre - rayon) de (CERCLE14, 1, lgsup, lginf, erreur, 0,
      -rayon);
    Envoi à (xcentre - rayon, ycentre) de (CERCLE14, 2, lgsup, lginf, erreur, -rayon,
      0);
    Envoi à (xcentre - rayon, ycentre - 1) de (CERCLE14, 2, lgsup, lginf, erreur + 4,
      -rayon, -1) ;
  FIN ;


PROCEDURE CERCLE14_CELLULE (I,J) ;

  DEBUT
    Reception (indicateur, lgsup, lginf, erreur, xrelatif, yrelatif) ;
    SI (xrelatif < 0) ALORS signex := -1 SINON signex := 1 ;
    SI (yrelatif < 0) ALORS signey := -1 SINON signey := 1 ;
    SI (indicateur = 1) ALORS
      DEBUT
        deplac1 := 1 ;
        deplac2 := 0 ;
        test := yrelatif ;
      FIN
    SINON
      DEBUT
        deplac1 := 0 ;
        deplac2 := signey ;
        test:= xrelatif ;
      FIN ;
    SI (test <> 0) ALORS
      SI (erreur <= lgsup) ALORS
        DEBUT
          SI (erreur >= lginf) ALORS
            Afficher ;
            Transmission (deplac1,deplac2) de (CERCLE14, indicateur, lgsup, lginf,
                  erreur + 8 * (deplac1 * xrelatif + deplac2 * yrelatif) + 4,
                  xrelatif + deplac1, yrelatif + deplac2))
        FIN
      SINON
        Transmission (-signex,-signey) de (CERCLE14, indicateur, lgsup, lginf,
                  erreur - 8 * (signex * xrelatif + signey * yrelatif - 1),
                  xrelatif - signex, yrelatif - signey))
    SINON
      Afficher ;
  FIN ;
```

Remarque :

Pour simplifier les calculs préliminaires effectué dans l'ordinateur hôte, nous avons remplacé les trois premières lignes par :

```
raycar := rayon2 ;
diam := 2 * rayon ;
lgsup := raycar + diam + 1 ;
lginf := raycar - diam + 1 ;
erreur := 4 * raycar ;
```

ALGORITHME DE TRACE DE CERCLES PARALLELE N°15

D.D.A.

```
PROCEDURE CERCLE15_HOTE ;

  DEBUT
    sinus := 1 / sqrt (rayon2 + 1) ;
    cosinus := sinus ;
    xrelatif := -rayon ;
    yrelatif := 0 ;
    Envoi à (xcentre - rayon, ycentre) de (CERCLE15, sinus, cosinus, xrelatif, yrelatif) ;
  FIN ;

PROCEDURE CERCLE15_CELLULE (I,J) ;

  DEBUT
    Reception (sinus, cosinus, xrelatif, yrelatif) ;
    SI (sinus = cosinus) ALORS
      DEBUT
        cosinus := -xrelatif * sinus ;
        Transmission (0,1) de (CERCLE15, sinus, cosinus, xrelatif, 1)) ;
        Transmission (0,-1) de (CERCLE15, -sinus, cosinus, xrelatif, -1)) ;
      FIN
    SINON
      SI (ARRONDI (yrelatif) <> 0) ALORS
        DEBUT
          edx := 0 ;
          edy := 0 ;
          ax := xrelatif ;
          ay := yrelatif ;
          TANT QUE (edx = 0) ET (edy = 0) FAIRE
            DEBUT
              dx := ax*cosinus + ay*sinus ;
              dy := ay*cosinus - ax*sinus ;
              edx := ARRONDI (dx) - ARRONDI (xrelatif) ;
              edy := ARRONDI (dy) - ARRONDI (yrelatif) ;
              ax := dx ;
              ay := dy ;
            FIN ;
          Transmission (edx,edy) de (CERCLE15, sinus, cosinus, ax, ay)) ;
        FIN ;
    Afficher ;
  FIN ;
```

Remarque :

Le tant que effectué pour éliminer les mouvements nuls peut être remplacé par un si, car l'application successive de deux rotations de l'angle élémentaire conduit nécessairement à deux points différents.

ALGORITHME DE TRACE DE CERCLES PARALLELE N°16

D.D.A. A REELS

```
PROCEDURE CERCLE16_HOTE ;

  DEBUT
    xrelatif := -rayon ;
    yrelatif := 0 ;
    Envoi à (xcentre-rayon, ycentre) de (CERCLE16, rayon, xrelatif, yrelatif) ;
  FIN ;

PROCEDURE CERCLE16_CELLULE (I,J) ;

  DEBUT
    Reception (rayon, xrelatif, yrelatif) ;
    SI ((xrelatif = -rayon) ET (yrelatif = 0)) ALORS
      DEBUT
        Transmission (0,1) de (CERCLE16, rayon, -rayon, -1) ;
        Transmission (0,-1) de (CERCLE16, rayon, -rayon, 1) ;
      FIN
    SINON
      DEBUT
        arry := ARRONDI (yrelatif) ;
        SI arry<> 0 ALORS
          edx := 0 ;
        edy := 0 ;
        ax := xrelatif ;
        ay := yrelatif ;
        SI (yrelatif>0) ALORS
          invrayon := -1/rayon
        SINON
          invrayon := 1/rayon ;
        TANT QUE (edx = 0) ET (edy = 0) FAIRE
          DEBUT
            ax := ax - invrayon*ay ;
            ay := ay + invrayon*ax ;
            edx := ARRONDI (ax) - ARRONDI (xrelatif) ;
            edy := ARRONDI (ay) - arry ;
          FIN ;
        Transmission (edx,edy) de (CERCLE16, rayon, ax, ay)) ;
      FIN ;
    Afficher ;
  FIN;
```

ALGORITHME DE TRACE DE CERCLES PARALLELE N°17

BRESENHAM A SYMETRIES

```
PROCEDURE CERCLE17_HOTE ;

  DEBUT
    erreur := 0 ;
    xrelatif := -rayon ;
    yrelatif := 0 ;
    Envoi à (xcentre-rayon, ycentre) de (CERCLE17, erreur, xrelatif, yrelatif) ;
    erreur := 1 ;
    yrelatif := -1 ;
    Envoi à (xcentre-rayon,ycentre-1) de (CERCLE17, erreur, xrelatif, yrelatif) ;
  FIN ;


PROCEDURE CERCLE17_CELLULE (I,J) ;

  DEBUT
    Reception (erreur, xrelatif, yrelatif) ;
    SI (xrelatif < 0) ALORS
      DEBUT
        ay := ABS (yrelatif) ;
        SI (yrelatif < 0) ALORS signey := -1 SINON signey := 1 ;
        erreurx := erreur + 2 * xrelatif + 1 ;
        erreury := erreur + 2 * ay + 1 ;
        erreurxy := erreurx + erreury - erreur ;
        Aerreurxy := ABS (erreurxy) ;
        SI (Aerreurxy <= ABS (erreury)) ALORS
          SI (Aerreurxy <= ABS (erreurx)) ALORS
            Transmission (1,signey) de (CERCLE17, erreurxy, xrelatif + 1,
              yrelatif + signey))
          SINON
            Transmission (1,0) de (CERCLE17, erreurx, xrelatif + 1, yrelatif))
        SINON
          Transmission (0,signey) de (CERCLE17, erreury, xrelatif,
            yrelatif + signey)) ;
      FIN ;
    Afficher ;
    Transmission (1,0) (AFFICHER, I-2*xrelatif);
  FIN ;
```

```
PROCEDURE AFFICHER_CELLULE (I,J) ;

  DEBUT
    Reception (x) ;
    SI (I=x) ALORS
      Afficher
    SINON
      Transmission (1,0) (AFFICHER,x);
  FIN
```

Remarque :

Cette procédure AFFICHER est différente dès lors qu'il existe un protocole de routage inter-cellulaire. Il ne s'agit plus alors que d'afficher le pixel de la cellule recevant la commande.

ANNEXE C

LES ALGORITHMES DE REMPLISSAGE

PREAMBULE

Les algorithmes que nous allons présenter seront moins détaillés que ceux relatifs au tracé de segments ou au tracé de cercles.

D'ailleurs un certain nombre d'entre eux utilisent des tracés de segments légèrement modifiés, nous n'en reprendrons pas l'étude ici, nous indiquerons simplement les modifications à y apporter.

D'autre part, certains traitements n'offrent pas d'intérêt dans le cadre de cette étude (par exemple : le tri d'une liste de manière séquentielle...), nous ne nous y attarderons pas.

Avant de se lancer dans la présentation des algorithmes, il faut se définir la façon de conserver les contours en mémoire.

La structure de données adoptée est des plus simple, puisqu'il s'agit de la liste des sommets du polygone.

Nous ajoutons deux contraintes à cette structure, afin de simplifier le remplissage :

- le parcours de cette liste nous donne un contour orienté dans le sens trigonométrique,
- le premier point de la liste est le point supérieur gauche du contour.

ALGORITHME DE REMPLISSAGE SEQUENTIEL N°1

TEST DE PARITE

```
DEBUT
   POUR (chaque ligne de l'image) FAIRE
      DEBUT
         Initialiser le compteur à zéro
         Partir du bord gauche de l'image
         TANT QUE (le bord droit n'est pas atteint) FAIRE
            DEBUT
               SI (le point est un point du contour) ALORS
                  Incrémenter le compteur
               SINON
                  SI (le compteur a une valeur impaire) ALORS
                     Le point est intérieur => Affichage
                  SINON
                     Le point est extérieur
               Passer au voisin de droite
            FIN
      FIN
FIN
```

Rappel :

Il s'agit ici de présenter le principe de l'algorithme. Le programme implanté tel quel conduit pour de nombreuses figures à un remplissage erroné.

ALGORITHME DE REMPLISSAGE SEQUENTIEL N°2

TEST DE PARITE SANS INSCRIPTION PREALABLE

```
DEBUT
   Lire le premier segment
   TANT QUE (le contour n'a pas été parcouru entièrement) FAIRE
      DEBUT
         TANT QUE (le segment suivant est horizontal) FAIRE
            Lire le segment suivant
         SI (le segment forme une pointe pleine avec le précédent) ALORS
            Prendre en compte les deux extrémités
         SINON
            SI (le segment forme une pointe creuse avec le suivant) ALORS
               Ne pas prendre en compte les extrémités
            SINON
               Prendre en compte la deuxième extrémité seulement
         Calculer les points du segment, avec le nombre d'extrémités adéquat,
         en conservant ceux utiles au remplissage (i.e. ceux qui succèdent ou précèdent
         un mouvement non-horizontal)
         Lire le segment suivant
      FIN
   Trier la liste par y, puis x croissants (i.e. pour un même y, ordonner par x
   croissants)
   TANT QUE (la liste n'est pas vide) FAIRE
      DEBUT
         Lire deux points consécutifs
         Tracer le segment horizontal les joignant
      FIN
FIN
```

ALGORITHME DE REMPLISSAGE SEQUENTIEL N°3

TEST DE PARITE APRES MARQUAGE

```
DEBUT
   POUR y variant de Ymin à Ymax FAIRE
      Initialiser intérieur à faux
      POUR x variant de Xmin à Xmax FAIRE
         DEBUT
            SI (le point (x,y) possède 1 comme marque) ALORS
               DEBUT
                  Complémenter intérieur
                  Afficher (x,y)
               FIN
            SINON
               SI ( ( intérieur est vrai ) ou ( la marque de (x,y) est 2 ) ) ALORS
                  Afficher (x,y)
         FIN
FIN
```

Xmin, Xmax, Ymin, Ymax *sont les coordonnées extrêmes du contour.*

Remarque :

Il s'agit simplement du remplissage, cela suppose l'existence d'un contour marqué convenablement. Ce marquage requiert, comme l'algorithme à inscription directe de la page précédente une analyse fine du contour.

ALGORITHME DE REMPLISSAGE SEQUENTIEL N°4

TEST DE PARITE : EXAMEN DU VOISINAGE

Procédure d'examen du voisinage de (x,y) :

```
DEBUT
   Initialiser les compteurs dessous et dessus à zéro
   Si  (x-1,y+1) appartient au contour)  ALORS
      Incrémenter dessus
   Si  ((x-1,y-1) appartient au contour)  ALORS
      Incrémenter dessous
   TANT QUE  (le point (x,y) appartient au contour)  FAIRE
      DEBUT
         SI  ( (x,y+1) appartient au contour
            et (x-1,y+1) n'appartient pas au contour )  ALORS
            Incrémenter dessus
         Si  ( (x,y-1) appartient au contour
            et (x-1,y-1) n'appartient pas au contour )  ALORS
            Incrémenter dessous
         Incrémenter x
      FIN
   Si  ( (x-1,y+1) appartient au contour)  ALORS
      Incrémenter dessus
   Si  ( (x-1,y-1) appartient au contour)  ALORS
      Incrémenter dessous
   Retourner les valeurs des compteurs dessus et dessous, et les coordonnées (x,y)
FIN
```

Procédure de remplissage proprement dit :

```
DEBUT
   POUR y variant de 0 à N  FAIRE
      Initialiser compteur à zéro
      Partir du bord droit
      TANT QUE  (le bord droit n'est pas atteint)  FAIRE
         DEBUT
            SI  ( (x,y) n'appartient pas au contour)  ALORS
               DEBUT
                  SI  (compteur est impair)  ALORS
                     Afficher  (x,y)
                  Passez au voisin de droite
               FIN
            SINON
               DEBUT
                  Examiner " VOISINAGE (x,y) "
                  SI  ( dessus et dessous sont à 1)  ALORS
                     Incrémenter compteur
                  SINON
                     SI  (dessus + dessous est différent de 0 ou 2)  ALORS
                        Le tracé est incorrect
               FIN
         FIN
FIN
```

ALGORITHME DE REMPLISSAGE SEQUENTIEL N°5

TEST DE PARITE : TABLEAU LOGIQUE

```
DEBUT
   Initialiser x et y à zéro
   TANT QUE  ( (x,y) n'appartient pas au contour)  FAIRE
      DEBUT
         SI  (x < Taille du réseau)  ALORS
            Incrémenter x
         SINON
            DEBUT
               Remettre x à zéro
               Incrémenter y
            FIN
      FIN
   Passer au point suivant
    (en parcourant le voisinage direct dans le sens trigonométrique à partir de la
    direction nord)
   TANT QUE  (le contour n'a pas été parcouru)  FAIRE
      DEBUT
         Chercher la direction du point suivant
         Regarder dans la table s'il faut conserver le point dans la liste
         Passer au point suivant
      FIN
   Trier la liste par y, puis x croissants (pour un même y, ordonner par x croissants)
   TANT QUE  la liste n'est pas vide  FAIRE
      DEBUT
         Lire deux points consécutifs
         Tracer le segment horizontal les joignant
      FIN
FIN
```

ALGORITHME DE REMPLISSAGE SEQUENTIEL N°6

POSITIONNEMENT PAR RAPPORT AUX ARETES

```
DEBUT
   POUR (chaque segment orienté du polygone) FAIRE
      DEBUT
         Initialiser à la première extrémité
         Calculer le premier déplacement
         TANT QUE (la deuxième extrémité n'est pas atteinte) FAIRE
            DEBUT
               Suivant le cas, s'il s'agit d'un déplacement :
                  Nord    : Complémenter de (0,y) à (x,y)
                  Est     : Ne rien faire
                  Sud     : Complémenter de (0,y) à (x-1,y)
                  Ouest   : Ne rien faire
               Calcul du déplacement suivant
            FIN
      FIN
FIN
```

Remarque :

Le calcul du déplacement dépend de la méthode employée, il peut s'agir notamment de l'algorithme de BRESENHAM.

ALGORITHME DE REMPLISSAGE SEQUENTIEL N°7

PROPAGATION LINEAIRE

```
DEBUT
   Empiler le point intérieur
   TANT QUE (la pile n'est pas vide) FAIRE
      DEBUT
         Dépiler un point (x,y)
         SI (le point n'appartient pas au contour) ALORS
            DEBUT
               Remplir_à_droite
               Remplir_à_gauche
               Explorer_au_dessus
               Explorer_en_dessous
            FIN
      FIN
FIN
```

Les procédures Remplir_à_droite et Remplir_à_gauche sont bâties sur le même schéma :

```
Remplir_à_droite :
DEBUT
   Mémoriser x (xsauve=x)
   TANT QUE  ( (le point (x,y) n'appartient pas au contour)
             et (le bord droit de l'écran n'est pas atteint) )  FAIRE
      DEBUT
         Afficher le pixel
         Passer au pixel de droite
      FIN
   Conserver l'extrémité droite (x-1)
   Réinitialiser x (x=xsauve)
FIN
```

Les procédures Explorer_au_dessus et Explorer_en_dessous sont semblables :

```
Explorer_au_dessus :
DEBUT
   SI (le sommet de l'image n'est pas atteint) ALORS
      DEBUT
         Mémoriser le point (xsauve=x,ysauve=y)
         Partir de l'extrémité gauche
         TANT QUE (l'extrémité droite n'est pas atteint) FAIRE
            DEBUT
               Parcourir les points du contour consécutifs situés sur la même ligne
               SI (l'extrémité droite n'est pas atteinte) ALORS
                  DEBUT
                     Empiler le point
                     Parcourir les points colinéaires n'appartenant pas au contour
                  FIN
            FIN
         Réinitialiser (x,y) à (xsauve,ysauve)
      FIN
FIN
```

ALGORITHME DE REMPLISSAGE SEQUENTIEL N°8

DIFFUSION

```
DEBUT
   Empiler le point intérieur
   TANT QUE (la pile n'est pas vide) FAIRE
      DEBUT
         Dépiler un point
         SI (le point n'appartient pas au contour) ALORS
            DEBUT
               Afficher le point
               Empiler ses voisins
            FIN
      FIN
FIN
```

Remarque :

C'est l'algorithme à la fois le plus court et le plus simple, mais son efficacité en mode séquentiel est peu satisfaisante.

LES ALGORITHMES PARALLELES

ANALYSE_GLOBALE

L' ANALYSE_GLOBALE du contour traite les singularités liées aux extrémités de segments et aux segments horizontaux.

La partie ANALYSE_GLOBALE du contour étant utilisée à maintes reprises dans les divers algorithmes, sous des formes différentes mais identiques dans le concept, nous allons décrire en détail la procédure générale.

```
ANALYSE_GLOBALE

DEBUT
   P1 := LIRE                    /* Lire le 1er point du contour */
   P2 := LIRE                    /* P1-P2 1er segment du contour */
   FIN := P1                     /* Mémorise le premier point */
   TRAITER ( 2, P1 , P2 )        /* Traitement du 1er segment avec ses deux
                                    extrémités puisqu'il s'agit nécessairement
                                    d'une pointe pleine */
   ORIENTATION#1 := 1            /* Signe de la pente du segment, égal à 1 puisqu'il
                                    s'agitdu point supérieur gauche */
   P1 := P2
   P2 := LIRE                    /* P1-P2 deuxième segment */
   TANT QUE (P1 <> FIN) FAIRE
      DEBUT
         /* Passer les segments horizontaux */
         TANT QUE (Y(P1) = Y(P2)) FAIRE
            DEBUT
               P1 := P2
               P2 := LIRE
            FIN
         ORIENTATION#2 := SGN ( Y(P2) - Y(P1) ) /* Orientation du segment */
         SI (ORIENTATION#1 = ORIENTATION#2) ALORS
            TRAITER (1, P1 , P2 )              /* Inflexion */
         SINON
            TRAITER (2, P1 , P2 )              /* Pointe */
         ORIENTATION#1 := ORIENTATION#2
         P1 := P2
         P2 := LIRE                            /* P1-P2 Segment suivant */
      FIN
FIN
```

Remarque :

*La distinction entre pointe pleine et pointe creuse intéressante en théorie, car elle retire des marques inutiles, n'est pas justifiée. En effet, cela entraîne des calculs plus nombreux lors de l'*ANALYSE_GLOBALE *qui annihilent totalement le gain en performances attendu.*

La procédure TRAITER possède comme premier paramètre le nombre d'extrémités à prendre en compte lors du traitement du segment.

Ce traitement consiste soit en la conservation des points intéressants dans une liste, soit en leur marquage, soit encore en l'exécution d'une complémentation à partir de ces points, cela en fonction de l'algorithme de remplissage auquel l'ANALYSE_GLOBALE est destinée.

Par la suite la procédure ANALYSE_GLOBALE ne sera plus détaillée. Pour les algorithmes qui y font appel, nous nous contenterons de préciser les fonctionnalités de la procédure TRAITER adaptée.

ALGORITHME DE REMPLISSAGE PARALLELE N°1

S.I.M.D.

```
HOTE
  DEBUT
    difx := XB - XA ;
    dify := YB - YA ;
    constante := difx*YA - dify*XA  ;
    POUR i := 0 A Taille_reseau FAIRE
      POUR j := 0 A Taille_reseau FAIRE
        Envoi à (i,j) de (REMPLISSAGE1, difx, dify, constante) ;
  FIN


CELLULE (i,j)

REMPLISSAGE1 :
  DEBUT
    Reception (difx, dify, constante) ;
    SI  ( (-dify*I+difx*J + constante) <= 0 )  ALORS
      Complementer ;
  FIN
```

ALGORITHME DE REMPLISSAGE PARALLELE N°2

MULTIPIPELINE

Remplissage direct basé sur le test de parité.

Cet algorithme utilise la procédure ANALYSE_GLOBALE définie précédemment.

La procédure TRAITER est dérivée étroitement des solutions séquentielles proposées pour le tracé de segments. Les points calculés correspondant aux extrémités des segments de remplissage sont conservés dans une liste.

```
HOTE
   DEBUT
      ANALYSE_GLOBALE              /* Construction de la liste */
      TRI de la liste par rapport aux Y
      TRI de la liste par rapport aux X /* Pour un Ydonné, ordonner par rapport à X */
      TANT QUE (la liste n'est pas vide) FAIRE
         DEBUT
            P1 = LIRE (liste)      /* Lire un point de la liste */
            P2 = LIRE (liste)      /* P1-P2 est un segment de remplissage */
            Envoi à ( 0 , Y(P1) ) de ( REMPLISSAGE2 , X(P1) , X(P2) )
         FIN
   FIN


CELLULE (i,j)

REMPLISSAGE2 :
   DEBUT
      Reception ( X1 , X2 )
      Transmission (1,0) de ( REMPLISSAGE2 , X1 , X2 )
      SI (X1<= i<=X2) ALORS
         Affichage
   FIN
```

Remarque :
La procédure REMPLISSAGE2 *correspond à l'affichage d'un segment horizontal.*

ALGORITHME DE REMPLISSAGE PARALLELE N°3

MULTIPIPELINE

Remplissage après marquage, basé sur le test de parité.

Le principe de l'algorithme est identique au précédent, si ce n'est que les points utiles au remplissage sont stocké par l'entremise des cellules. Cela permet de distribuer le travail au réseau.

L'ANALYSE_GLOBALE définie par ailleurs est ici encore utilisée, avec une procédure de traitement différente :

TRAITER correspond dans ce cas à l'affichage des segments du contour, avec le marquage des extrémités des segments de remplissage. Suivant l'orientation, le point marqué est soit le premier, soit le dernier rencontré sur la ligne.

```
HOTE
   DEBUT
      ANALYSE_GLOBALE              /* Tracé du contour et marquage */
      INTERIEUR := FAUX
      POUR Y de Ymin à Ymax FAIRE
         Envoi à (0,Y) de ( REMPLISSAGE3 , INTERIEUR )
   FIN
```

Outre le tracé du contour avec marquage, étroitement dérivé du tracé de segment (ANNEXE A, algorithme N°2), les cellules ont à afficher les segments de remplissage.

```
CELLULE (i,j)

REMPLISSAGE3 :
   DEBUT
      Reception ( INTERIEUR )
      SI (marque = 2) ALORS
         Transmission (1,0) de ( REMPLISSAGE3 , INTERIEUR )
      SINON
         Transmission (1,0) de ( REMPLISSAGE3 , NON (INTERIEUR) )
      SI (INTERIEUR) ALORS
         Affichage
   FIN
```

```
CELLULE (i,j)

PTSUP :
  DEBUT
    SI (la cellule ne fait pas partie du contour) ALORS
      Transmission (1,0) de ( PTSUP )          /* Continuer la recherche */
    SINON
      DEBUT
        /* Chercher le point suivant parmi les quatres possibles : SO, S, SE, E */
        Cell_Vois := O
        Cell_Succ := FAUX
        TANT QUE (Cell_Succ = FAUX) FAIRE
          DEBUT
            Cell_Vois := SUCC ( Cell_Vois ) /* Examen de la cellule suivante */
            Emis_Rec (Cell_Vois) de (SUIVI,i,j) (Cell_Succ) /*Interrogation*/
          FIN
        /* Traiter la cellule */
        SI ( Cell_Vois = NE ) ALORS
          ERREUR       /* Cas impossible car point supérieur gauche */
        SINON
          Marquer      /* Le 1er point est toujours marqué */
      FIN
  FIN
```

NB : La fonction SUCC est une relation d'ordre sur les scalaires représentants les directions : (N, NO, O, SO, S, SE, E, NE) => SUCC(O)=SO, SUCC(NE)=N...

```
SUIVI :
  DEBUT
    Reception ( X , Y )
    SI (la cellule n'appartient pas au contour) ALORS
      Retourner ( FAUX )
    SINON
      DEBUT
        Retourner ( VRAI )
        /* Chercher le successeur */
        Cell_Vois := Emettrice     /* Connue grâce au port de réception */
        Cell_Succ := FAUX
        TANT QUE (Cell_Succ = FAUX) FAIRE
          DEBUT
            Cell_Vois := SUCC ( Cell_Vois ) /* Examen de la cellule suivante */
            Emis_Rec (Cell_Vois) de (SUIVI,i,j) (Cell_Succ) /*Interrogation*/
          FIN
        /* Calcul de la marque à l'aide du tableau logique */
        Marquer?
      FIN
  FIN
```

ALGORITHME DE REMPLISSAGE PARALLELE N°4

MULTIPIPELINE

Remplissage d'un contour préinscrit par la méthode du test de parité, avec traitement des singularités à l'aide d'un tableau logique.

Par hypothèse le tableau logique est supposé connu de chacune des cellules, il fait partie intégrante du micro-programme.

```
HOTE
  DEBUT
    Envoi ( 0 , Ymax ) de ( PTSUP )      /* Analyse et marquage du contour */
    Contrôle                             /* Attendre la fin du marquage */
    INTERIEUR := FAUX
    POUR Y de Ymin à Ymax FAIRE
      Envoi à (0,Y) de ( REMPLISSAGE4 , INTERIEUR )
  FIN
```

Pour connaître le point suivant il faut interroger les cellules voisines, il y a une séquence question-réponse qui ne va pas sans poser de problèmes de synchronisation.

Un nouveau protocole de communication est donc défini :

- pour la cellule demandeuse :

  Emis_Rec (Direction) de (Commande) (Reponse)
  où Direction est N, NO, O, SO, S, SE, E ou NE
  Commande est la commande paramétrée transmise, et Reponse est un booléen

- la cellule interrogée doit répondre dès que possible par:

  Retourner (Reponse)
  où Reponse est le booléen envoyé à la cellule émettrice

Attention : Ce type de protocole doit être manipulé avec précautions afin de ne pas engendrer d'inter-blocages.

```
REMPLISSAGE4 :
  DEBUT
    Reception ( INTERIEUR )
    SI (MARQUE) ALORS
      Transmission (1,0) de ( REMPLISSAGE4 , INTERIEUR )
    SINON
      Transmission (1,0) de ( REMPLISSAGE4 , NON (INTERIEUR) )
    SI (INTERIEUR) ALORS
      Affichage
  FIN
```

ALGORITHME DE REMPLISSAGE PARALLELE N° 5

SUIVI DE CONTOUR

Remplissage par examen de la position relative des points par rapport à chaque arête du contour.

```
HOTE
  DEBUT
    ANALYSE_GLOBALE
  FIN
```

La procédure TRAITER utilisée dans l'ANALYSE_GLOBALE consiste à émettre une commande de tracé de segment vers le réseau. Ce tracé de segment est un peu particulier : pour chaque ligne de balayage coupant le segment il y a émission d'une commande de complémentation à droite. Pour chaque ligne le point à partir duquel a lieu la complémentation est celui succédant ou précédant un mouvement non-horizontal.

Les modifications à apporter à l'algorithme de BRESENHAM parallélisé (cf ANNEXE A : algorithme N°7) sont donc minimes.

```
CELLULE (i,j)

REMPLISSAGE5 :
  DEBUT
    Transmission (1,0) de (REMPLISSAGE5 ) ;
    Complementer ;
  FIN
```

Remarque :
La procédure REMPLISSAGE5 *complémente directement le pixel associé à la cellule.*

ALGORITHME DE REMPLISSAGE PARALLELE N° 6

MULTIPIPELINE

Remplissage par examen de la position relative des points par rapport à chaque arête du contour.

```
HOTE
  DEBUT
    ANALYSE_GLOBALE
  FIN
```

La procédure TRAITER, utilisée dans l'ANALYSE_GLOBALE est simplement une procédure de tracé de segments (cf ANNEXE A, algorithme N°2) dans laquelle on a modifié le seuil d'affichage afin de complémenter les pixels situés à droite du segment. En fait, plutôt que de comparer la valeur absolue de l'erreur commise au seuil d'affichage, on complémente les points pour lesquels l'erreur est négative.

Remarque :
Le mode de fonctionnement étant de type multipipeline, la complémentation s'effectue sur une variable interne de la cellule. Une vague supplémentaire réalise alors l'affichage proprement dit. Cette vague peut d'ailleurs être commune à plusieurs remplissages.

CELLULE (i,j)

```
REMPLISSAGE6 :
  DEBUT
    Transmission (1,0) de (REMPLISSAGE6 ) ;
    SI (VAR_INT = VRAI) ALORS
      Afficher;
  FIN
```

ALGORITHME DE REMPLISSAGE PARALLELE N°7

PROPAGATION DIRIGEE

Remplissage à partir d'un point intérieur au contour.

```
HOTE
  DEBUT
    Envoi à la cellule intérieure de ( REMPLISSAGE7 )
  FIN
```

Les procédures exécutées par les cellules sont :

La procédure REMPLISSAGE7 se limite à la recherche d'un point du contour :

CELLULE (i,j)

```
REMPLISSAGE7 :
  DEBUT
    SI (la cellule n'appartient pas au contour) ALORS
      Transmission (1,0) de ( REMPLISSAGE7 ) /* Continuer la recherche */
    SINON
      Transmission (-1,0) de ( REMPH , i-1 ) /* Commencer l'affichage */
    FIN
  FIN
```

La procédure REMPH affiche tous les points parcourus jusqu'à rencontrer un point du contour, où alors elle émet deux commandes d'examen sur les lignes voisines.

```
REMPH :
  DEBUT
    Reception ( X )
    SI (la cellule n'appartient pas au contour) ALORS
      DEBUT
        /* Continuer l'affichage */
        Transmission (-1,0) de ( REMPH , X )
        Afficher
      FIN
    SINON
      DEBUT
        /* Envoyer les commandes d'examen */
        Transmission (1,1) de ( EXAMEN , X )
        Transmission (1,-1) de ( EXAMEN , X )
      FIN
  FIN
```

Remarque : Le paramètre X transmis de cellule en cellule est utilisé pour borner l'examen des lignes voisines.

La procédure EXAMEN est calquée sur le programme séquentiel, si ce n'est qu'au lieu d'empiler un point, on commande l'exécution de la procédure REMPLISSAGE7 sur la cellule.

```
EXAMEN :
   DEBUT
      Reception ( BORNE , INTERIEUR )
      SI (i <= BORNE) ALORS
         SI  (le point appartient au contour)  ALORS
            Transmission ( EXAMEN , BORNE , FAUX )
         SINON
            DEBUT
               Transmission ( EXAMEN , BORNE , VRAI )
               SI (INTERIEUR = FAUX) ALORS
                  REMPLISSAGE7     /* Execution de la commande */
            FIN
   FIN
```

ALGORITHME DE REMPLISSAGE PARALLELE N°8

DIFFUSION

Remplissage à partir d'un point intérieur.

```
HOTE

   DEBUT
      Envoi à la cellule intérieur de ( REMPLISSAGE8 )
   FIN


CELLULE (I,J)

REMPLIR :
   DEBUT
      SI  (le pixel de la cellule est éteint)  ALORS
         Diffusion de ( REMPLISSAGE8 )
      Affichage
   FIN
```

Remarque :

L'instruction de Diffusion *communique la commande à toutes les voisines de la cellule.*

Le test sur l'appartenance au contour est fait directement sur le pixel, ce qui permet d'éviter de traiter une nouvelle fois les pixels déjà traités.

BIBLIOGRAPHIE

BIBLIOGRAPHIE

[ABJ 85]
D. AYALA, P. BRUNET, R. JUAN & I. NAVAZO
"Object representation by means of non-minimal division quadtrees and octrees"
ACM Transactions on graphics Vol 4 N°1 pp41-59 01/85

[ACK 81]
B.D. ACKLAND & N.H. WESTE
"The edge flag algorithm_A fill method for raster scan display"
IEEE Transaction on computer Vol C-30 N°1 01/81

[AKN 85]
J.V. AKEN & M. NOVAK
"Curve drawing algorithms for raster displays"
ACM Transactions on graphics Vol 4 N°2 04/85

[ARO 86]
H.R. ARABNIA & M.A. OLIVIER
"Fast operations on raster images with SIMD machines architectures"
Computer graphics forum Vol 5 pp179-188 1986

[ARO 87]
H.R. ARABNIA & M.A. OLIVIER
"Arbitrary operations of raster images with SIMD machines architectures"
Computer graphics forum Vol 6 pp3-12 1987

[ATA 89]
A. ATAMENIA
"Architectures cellulaires pour la synthèse d'images"
Thèse de Doctorat USTL FA Lille Juin 1989

[BAT 80]
K.E. BATCHER
"Design of a massively parallel processor"
IEEE Transactions on computers Vol C-29 N°9 pp836-840 07/80

[BHR 88]
S.K. BHASKAR & A. ROSENFELD
"Parallel processing of regions represented by linear quadtrees"
Computer vision, graphics & image processing
Vol 42 N°5 pp 371-380 06/88

[BIS 85]
S.N. BISWAS & B.B. CHAUDHURI
"On the generation of discrete circular objects and their properties"
Computer Vision, Graphics and Image processing Vol C-19 03/85

[BRE 77]
J. BRESENHAM
"A linear algorithm for incremental digital display of circular arcs"
Graphics and image processing Vol 20 N°2 02/77

[BRF 79]
K.E. BRASSEL & R. FEGEAS
"An algorithm for shading of regions on vector display devices"
SIGGRAPH Proceedings pp 126-133 1979

[CAR 77]
C. CARREZ
"Control of an incremental plotter by a micro processor"
Publication N°94 du laboratoire de calcul de Lille 1977

[CHA 84]
B. CHAZELLE & J. INCERPI
"Triangulating and shape complexity"
ACM transactions on graphics Vol 13 N°2 pp135-152 04/84

[CHR 80]
A. CHRYSSAFIS
"Anti-aliasing on computer generated images"
Computer Graphics Forum N°5 pp 125 1980

[CML 87]
D.M. CHUARULLI, R.G. MELHEM & S.P. LEVITAN
"Using coincident optical pulses for parallel memory addressing"
Computer Vol 20 N°12 12/87

[COD 68]
E.F. CODD
"Cellular automata"
Academic press New York 1968
MARI 87 CESTA pp203-210 05/87

[DAN 70]
E. DANIELSON
"Incremental curve generation"
IEEE Transactions on computers Vol C-19 N°9 09/70

[DUN 83]
M.R. DUNLAVEY
"Efficient polygon-filling algorithms for raster displays"
ACM Transactions on Graphic Vol 2 N°4 10/83

[DUR 83]
Ph. DURIF
"Etude d'une machine parallèle de synthèse d'images à découpage par objets"
Thèse de Docteur 3ème cycle Lille 08/12/83

[FIE 83]
D.E. FIELD
"Algorithms for drawing simple geometric objects on raster devices"
University Microfilms International 1983

[FIE 83]
D.E. FIELD
"Incremental linear interpolation"
ACM Transactions on graphics Vol 4 N°1 01/85

[FOM 84]
A. FOURNIER & D. MONTUNO
"Triangulating simple polygons and equivalent problems"
ACM Transactions on Graphic Vol 3 N°2 04/84

[FOU 88]
J .FOUCHART
"La lumière sur les écrans plats"
Micro-systèmes N°91 pp89-99 10/88

[FPW 87]
G. FARIN & B. PIPER & A.J. WORSEY
"The octant of a sphere as a non degenerate triangular bezier patch"
Computer aided geometric design Vol 4 N°4 pp329-332 12/87

[FRE 61]
H. FREEMAN
"On the encoding of arbitrary geometric configurations"
IEEE Transactions on computers Vol 10 N°2 06/61

[FRE 80]
H. FREEMAN
"Tutorial and selected readings in interactive computer graphics"
IEEE Computer society 1980

[FRI 83]
P. FRISON, F. ANDRE & P. QUINTON
"Techniques de conception et de programmation d'architectures systoliques"
IRISA Publication interne N°196 03/87

[FUC 81]
H. FUCHS & J. POULTON
"Pixel-planes : a V.L.S.I oriented design for raster graphics engine"
VLSI design N°36 pp20-28 1981

[FUC 85]
H. FUCHS & Al
"Fast spheres, shadows, textures, transparencies and image enhancements in Pixel-plane"
Computer graphics Vol 19 N°3 pp111-120 06/85

[FUC 88]
H. FUCHS & Al
"Pixel-planes 4 : a summary"
Advances in computer graphics hardware II Springer-Verlag 1988

[FWA 87]
J.A.B. FORTES & B.W. Wa
"Systolic arrays_From concept to implementation"
Computer Vol 20 N°7 pp12-17 07/87

[GAL 87]
G. GARCIA & J.F. LE CORRE
"Transformations géométriques d'une image représentée par un quadtree"
MARI 87 COGNITIVA 87 05/87

[GHL 88]
J.B. GRUN, B. HORNELAGE & R. LEVY
"L'ordinateur optique"
La recherche N°204 11/88

[GOP 88]
J-P. GOURRET & J. PAILLE
"Irregular polygon fill using contour encoding"
Computer Graphics Forum Vol 6 N°4 pp 317-325 1988

[GOU 80]
H. GOURAUD
"Continuous shading of curved surfaces"
Interactive Computer Graphics pp 302-308 1980

[HAN 86]
I. HANNEQUIN
"Techniques d'antialiasing en infographie"
Mémoire de D.E.A. Lille 1986

[HEG 85]
G. HEGRON
"Synthèse d'image : algorithmes élémentaires"
Dunod informatique pp20-25 1985

[HEM 87]
F. HEMERY
"Etude d'un réseau cellulaire multipipeline. Simulation sur Transputer en langage OCCAM"
Mémoire de DEA Lille 1987

[HER 83]
S. HERTEL & K. MELHORN
"Fast triangulation of simple polygons"
Proceedings of the 1983 international FCT conférence pp207-218 08/83

[HFU 83]
K. HWANG & K. FU
"Integrated computer architectures for image processing"
Computer Vol 16 N°1 pp51-60 01/83

[HJL 73]
B.D. HOLM & B.W. JORDAN & W.J. LENNON
"An improved algorithm for the generation of non-parametrics curves"
IEEE Transactions on computers Vol C-22 N°12 12/73

[ILR 83]
M.D. Mac ILROY
"Best approximate circles on integer grids"
ACM Transactions on graphics Vol 2 N°4 10/83

[ISO 73]
N.T. ISON
"An algorithm to shade a plot"
Computer Graphics Vol 7 N°3 1973

[KID 83]
M. KIDODE
"Image processing machines in japan"
Computer Vol 16 N°1 pp68-80 01/83

[KIL 87]
KILGOUR
"Unifying vector and polygon algorithms for scan conversion and clipping"
EUROGRAPHICS 1987

[KKS 79]
T. KAMAE & T. KOBAYASHI & V. SUENAGA
"A high speed algorithm for the generation of straight line and circular arcs"
IEEE Transactions on computers Vol C-28 N°10 10/79

[KUN 82]
S.Y. KUNG & Al
"Wave front array processor: language, architecture and applications"
IEEE Transactions on computers Vol C-31 N°11 11/82

[KUN 87]
S.Y. KUNG & Al
"Wave front array processors_ Concept to implementation"
Computer Vol 20 N°7 pp18-33 07/87

[LAN 83]
J.M. LANE
"An algorithm for filling regions on graphics display devices"
ACM Transactions on Graphics Vol 2 N°3 07/83

[LEP 88]
E. LEPRETRE
"Line-drawing algorithms for cellular architectures"
Publication Interne LIFL USTL FA 1988

[LIE 88]
M.L.P. LIEROP & Al
"Line rasterization algorithms that satisfy the subset line property"
Computer vision, Graphics and Image processing
Vol 41 N°2 pp210-228 02/88

[LIT 79]
W.D. LITTLE & R. HEUFT
"An area shading graphics display system"
IEEE Transactions on Computers Vol C-28 N°7 07/79

[LMA 86]
E. LEPRETRE & M. MERIAUX & A. ATAMENIA
"A transputer based architecture for graphics"
7th Occam Users Group Workshop 1986

[LOC 80]
M. LOCEFF
"The line"
Computer pp57 06/80

[LUC 77]
M. LUCAS
"Contribution à l'étude des techniques de communication graphique avec un ordinateur"
Thèse d'état IMAG 1977

[LUC 82]
M. LUCAS
"La réalisation de logiciels graphiques interactifs"
Ed. Eyrolles 1982

[MAR 82]
F. MARTINEZ
"Vers une approche systématique de la synthèse d'images"
Thèse d'état INP 11/82

[MAZ 88]
G. MAZARE
"Une nouvelle architecture cellulaire pour la reconstruction d'images"
PIXIM 88 Ed Hermès 10/88

[MBM 87]
P. MARTIN & T. BONNET & Y. MATHIEU
"Circuit systolique pour la synthèse d'images"
MARI 87 COGNITIVA 87 05/87

[MER 84]
M. MERIAUX
"Contributions à l'imagerie informatique : Aspects algorithmiques et architecturaux"
Thèse de docteur es sciences mathématiques Lille 07/84

[MOR 76]
P. MORVAN & M. LUCAS
"Images et ordinateurs, introduction à l'infographie interactive"
Ed Larousse université série informatique 1976

[NEU 66]
J. VON NEUMANN
"Theory of self reproducing automata"
University of illinois press, urbana, 1966

[NWS 81]
W.M. NEWMAN & R.F. SPROULL
"Principles of interactive computer graphics"
Second Edition, Mac Graw Hill International 1981

[OUL 88]
D. OULD BRAHIM
"Etude d'un nouveau modéle de primitive pour la description des images numériques : Le macro-pixel"
Thèse de docteur ingénieur Lille 09/88

[PAB 84]
D. PANOGIOTOPOULOS & N. BOURBAKI
"The VLSI design of a two dimensional processing array"
Microprocessing & Microprogramming Vol 14 pp125-132 1984

[PAV 82]
T. PAVLIDIS
"Algorithms for graphics and image processing"
Springer-Verlag 1982

[PEL 85]
M. PELERIN
"Synthèse d'images et parallélismes : algorithmes et architectures"
Thèse de Docteur 3ème cycle Lille 03/01/85

[PER 88]
B. PEROCHE & Al
"La synthèse d'images"
Hermès traité des nouvelles technologies 1988

[PHO 75]
PHONG B.T.
"Illumination for computer generated pictures"
Communications ACM Vol 18 N°6 06/75

[PIE 86]
L. PIEGL
"The sphere as a rational bezier surface"
Computer aided geometric design Vol 3 N°1 pp45-52 05/86

[PIE 88]
F. PIERRE & Al
"Description d'une machine de traitement d'images temps réel : l'approche multiparallèlisme"
PIXIM 88 Ed Hermès 10/88

[POT 83]
J.L. PLOTTER
"Image processing on the massively parallel processor"
Computer Vol 16 N°1 pp62-67 01/83

[PRD 84]
K. PRESTON & M.J.B. DUFF
"Modern cellular automata"
Plenum press, New-york and London 1984

[REV 88]
J.P. REVEILLES
"Les paliers des droites de BRESENHAM"
PIXIM 88 Ed Hermès 10/88

[RGZ 85]
A. REICHART, P. GARDA & B. ZAVIDOVIQUE
"Seeing retinas"
MARI 87 COGNITIVA 87

[ROS 74]
A. ROSENFELD
"Digital straight line segments"
IEEE Transactions on Computer Vol C-25 N°12 12/74

[ROS 83]
A. ROSENFELD
"Parallel image processing using cellular arrays"
Computer Vol 16 N°1 pp14-20 01/83

[SCH 78]
B. SCHACHTER
"Decomposition of polygons into convex sets"
IEEE Transactions on Computers Vol C-28 N°7 07/79

[SEI 85]
C.L. SEITZ
"The cosmic cube"
Communications of ACM N°1 Vol 28 01/1985

[SHS 87]
C.A. SHAFFER & H. SAMET
"Optimal quadtree construction algorithms"
Computer vision, graphics & image processing Vol 37 pp 402-419 1987

[SMI 79]
A.R. SMITH
"Tint fill"
Proceedings SIGGRAPH ACM pp276-283 08/79

[STA 74]
C.D. STAMOPOULOS
"Parallel algorithm for joining two points by a straight line segment"
IEEE Transactions on computers Vol C-23 N°6 pp642-647 06/74

[STA 75]
C.D. STAMOPOULOS
"Parallel image processing"
IEEE Transactions on computers Vol C-24 N°4 pp424-429 04/75

[TAN 88]
G.Y. TANG & B. LIEN
"Region filling with the use of this discrete Green theorem"
Computer vision , graphics and image processing Vol 42 N°3 06/88

[THP 88]
T. THEOARIS & I. PAGE
"Incremental polygon rendering on a SIMD processor array"
Computer graphics forum Vol 7 pp331-341 1988

[WOL 86]
S. WOLFRAM
"Theory and applications of cellular automata"
World Scientific publishing 1986

[WOO 85]
T.C. WOO & S.Y. SHIN
"A linear time algorithm for triangulating a point visible polygon"
ACM transactions on computers Vol C-28 N°1 pp60-70 01/85

Algorithmes parallèles et architectures cellulaires pour la synthèse d'images

La synthèse d'images est une des disciplines de l'informatique les plus gourmandes en puissance de calcul. En effet, si les traitements à appliquer sont relativement simples, le temps qui leur est imparti est très limité eu égard à l'interactivité du système.

La répétition des calculs élémentaires et leur relative indépendance, conduit à opter en faveur d'une parallélisation massive du traitement. L'approche cellulaire, avec les machines-pixels, est une des façons d'aborder le problème.

Ce travail a pour objectif de définir les meilleures solutions architecturales en se fondant sur la parallélisation des algorithmes de base de la synthèse d'images (tracé de segments, tracé de cercles, remplissage...).

Pour chacun des problèmes de base, une dizaine d'algorithmes parallèles sont développés conduisant à six architectures différentes. Leurs performances relatives sont alors comparées afin de retenir les meilleures d'entre elles.

Une synthèse finale nous permet de définir quelles solutions sont réellement envisageables et quelles sont les perspectives de développement.

Mots-clés :
Synthèse d'images, Parallélisme, Architectures cellulaires,
Tracé de segments, Tracé de cercles, Remplissage