USTL FLANDRES ARTOIS

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

LABORATOIRE D'INFORMATIQUE FONDAMENTALE DE LILLE

N° d'ordre: 551

THESE

présentée à

L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE
FLANDRES ARTOIS

pour obtenir le titre de

DOCTEUR en **INFORMATIQUE**

par

Ahmed SERHROUCHNI

ETUDE D'ARCHITECTURES PARALLELES A BASE DE PROCESSEURS GRAPHIQUES SPECIALISES

THESE NON CORRIGÉE

Thèse soutenue le 28 Juin 1990 devant la commission d'Examen

Membres du jury

Mr. V. CORDONNIER	Président et Directeur de thèse
Mr. G. FONTENIER	Rapporteur
Mr. B. TOURSEL	Rapporteur
Mr. M. Mériaux	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:

Monsieur Vincent CORDONNIER, Professeur à l'Université des Sciences et Techniques de Lille Flandres-Artois, qui a bien voulu présider ce jury, m'a proposé ce sujet et m'a judicieusement conseillé tout au long de l'évolution de ce travail.

Messieurs Guy FONTENIER, Professeur à Télécom Paris, et Bernard TOURSEL, Professeur à l'Ecole Universitaire d'Ingénieurs de Lille, rapporteurs de cette thèse, qui m'ont fait l'honneur de consacrer leur temps à l'examen de ce travail,

Monsieur Michel MERIAUX, Chargé de recherche au C.N.R.S., qui m'a fait l'honneur de participer à ce jury, et qui n'a pas hésité à me fournir conseils et soutien scientifique,

J'ai apprécié l'ambiance agréable entretenue par les membres du L.I.F.L., et je remercie Monsieur Henri GLANC qui a reproduit avec diligence ce document.

Enfin, mes remerciements vont à mes parents, mes frères et soeurs, ma cousine Bouchra, mon cousin Boucif, et mes amis(es) pour leur soutien.

PLAN

PLAN

CHAPITRE I

GENERALITES

SOMMAIRE

I - INTRODUCTION:

Un système graphique peut être défini comme étant l'ensemble des moyens informatiques mis en oeuvre afin d'assurer le traitement, le stockage et la visualisation des informations concernant les "images", et les "dessins" appelés aussi informations graphiques. Cette appelation est prise ici dans son sens le plus large, car une "image" est définie sous des formes codées différentes depuis sa création (par l'intermédiaire d'un clavier alpha-numérique par exemple) puis son passage par les différentes chaînes de traitement (sous d'autres formes codées éventuellement) jusqu'à sa visualisation (sous forme d'intensités lumineuses sur un écran de télévision par exemple).

II - SYSTEME GRAPHIQUE:

On peut distinguer deux types de systèmes graphiques:

* système de synthèse d'images:

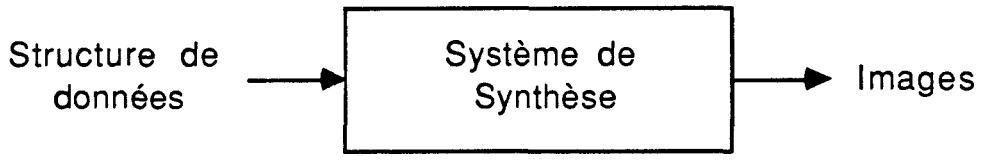


Fig 1

L'image constitue pour ces systèmes une finalité.

* système de traitement d'images:

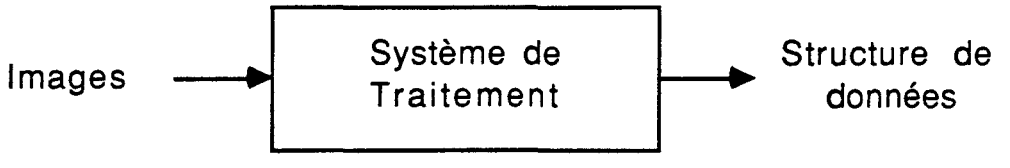


Fig 2

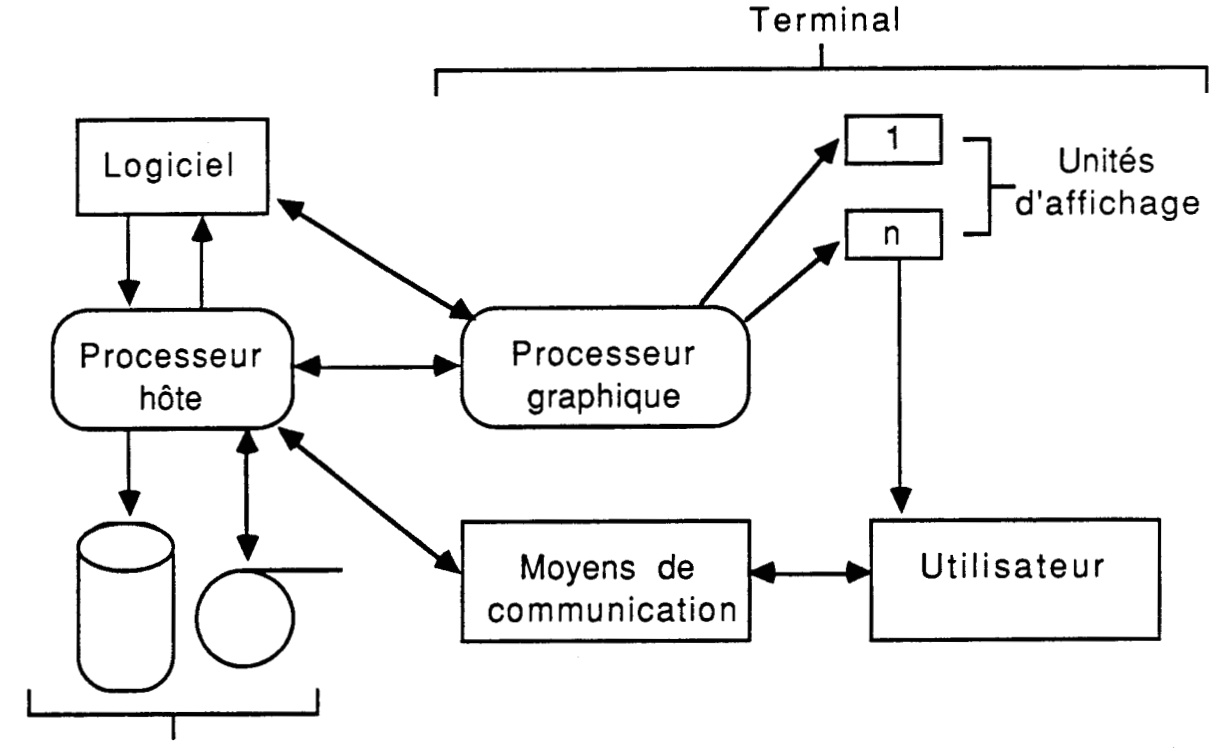
L'image constitue pour ces systèmes un support d'information.

Nous axons ce travail sur la synthèse et plus précisement sur les aspects architecturaux de cette synthèse.

III - SYSTEME DE SYNTHESE D'IMAGES:

A - ASPECT MATERIEL:

La structure d'un système de synthèse d'images est la suivante ([MER 79], [MOR 76]):



STRUCTURE D'UN SYSTEME DE SYNTHESE D'IMAGES

Fig 3

* un processeur hôte

* un terminal pour la visualisation des images

* des unités de stockage pour mémoriser les informations du système graphique

* des logiciels pour gérer les programmes utilisateurs et offrir des fonctions et des primitives de programmation

* des unités de dialogue pour assurer la communication entre utilisateur et le système graphique

1 - Processeur hôte:

Il permet de décharger le processeur graphique des traitements informatiques plus classiques tels que: gstion de bases de données, compilation, gestion de fichiers, ...etc

2 - Terminal:

Il se compose des unités d'affichage et d'un processeur graphique.

a - Unités d'affichage:

Ce sont les moyens physiques d'affichage des images ou dessins comme les imprimantes, les tables traçantes, les écrans à tubes cathodiques ...etc

b - Processeur graphique:

C'est la partie intelligente du terminal. Il s'occupe de:

* La gestion de la mémoire de trame (cas d'un balayage télévision) ou de la liste de visualisation (cas d'un affichage en mode cavalier)

* L'interprétation des commandes issues de l'utilisateur et des unités de traitement, et de l'exécution de ces dernières sur l'image à visualiser

* Eventuellement du transfert entre les mémoires de rafraîchissement et les unités d'affichage (cas de la table traçante: initialisation des programmes canaux assurant la sortie sur table traçante ...)

C'est pourquoi il est nécéssaire que le processeur graphique soit trés rapide, performant et spéçialisé, c'est l'une des raisons pour lesquelles plusieurs applications actuelles utilisent des processeurs microprogrammables.

3 - Unités de stockage:

Ce sont les mémoires de masse du système. Elles permettent la mémorisation d'ensembles importants d'informations (bibliothèques de programmes, compilateurs, bases de données,...etc). Parmi celles là on trouve les disques, les bandes magnétiques, disquettes, vidéodisques, bandes magnétiques vidéo ...

4 - Logiciels graphiques:

Ils regroupent toutes les fonctions programmées du système.

5 - Unités de communication:

Ils permettent le dialogue par l'intermédiaire de fonctions de programmation et de commande (commandes de chargement et d'exécution de programmes, mise au point, ...) et de transmissions de données (transmissions de coordonées ...).

Ces systèmes sont soit:

* des moyens d'entées-sorties classiques (claviers alpha-numériques, imprimantes, unités de disquettes, ...)

* des moyens spécifiques à l'informatique graphique, comme:

a - Stylo , Phostyle:

Ils détectent le passage du spot lors du balayage et fournissent ses coordonnées ou permettent d'identifier un objet.

b - Manche à balai , Boule roulante:

Ils permettent de désigner un point sur l'écran. Ils sont associés à un marqueur (flèche ou réticule) sur l'écran.

c - Clavier de fonctions:

C'est un boîtier de touches relié au système graphique. Chaque touche peut être programmée pour réaliser une fonction spécifique.

d - Tablette à numériser , Digitaliseur:

Ils permettent la transmission, au processeur, des coordonnées du crayon avec lequel on dessine.

B - ASPECT FONCTIONNEL:

Toute machine de synthèse d'images est constituée de trois modules principaux (d'après la classification de Martinez [MAR 86]): un module pilote, un module graphique, un module vidéo. Le premier appartient au monde des logiciels, le deuxième à celui des interfaces et le dernier à celui de la vidéo. Le passage d'un monde à l'autre s'effectue par l'intermédiaire de deux frontières que sont le bus de l'ordinateur d'une part, la mémoire de trame d'autre part.

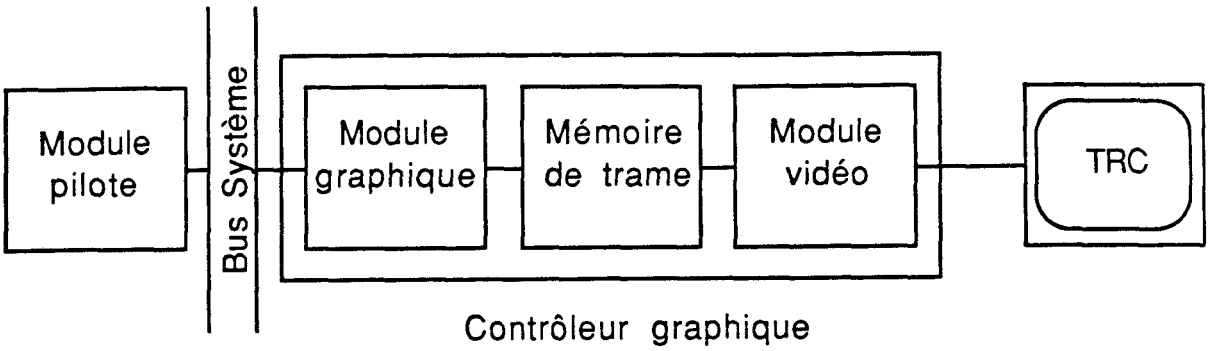


Fig 4

Nous allons décrire, brievement, les fonctionnalités de chacun des trois modules.

FONCTIONNALITES DES MODULES:

1 - MODULE PILOTE:

Il prend en charge la gestion des structures de données, les calculs géométriques, la projection perspective dans le plan de l'image, le coupage des objets et les calculs permettant de détérminer la couleur des points affichés. Il s'occupe également du pilotage du module graphique. Ce module posséde au moins un processeur d'usage général, mais il peut en contenir plusieurs qui se répartissent alors les tâches générales et les fonctions purement graphiques.

2 - MODULE GRAPHIQUE:

Inversement au module pilote, le module graphique traite essentiellement des objets 2D. Il s'occupe de la génération des objets graphiques élémentaires (droites, cercles, caractères, remplissage de polygones, ...) dans la mémoire de trame, de l'exécution d'algorithmes de base (algorithme du tampon de

profondeur pour l'élimination des parties cachées, algorithme d'interpolation bilinéaire de Gouraud pour la simulation de surfaces gauches).

3 - MODULE VIDEO:

Mis à part la conversion numérique analogique, la génération des signaux de synchronisation et la gestion des priorités éventuelles entre plusieurs images. Les traitements, réalisés par ce module, doivent être effectués pixel par pixel, au rythme du balayage vidéo (moins de 74 ns par pixel pour une image compatible vidéo), ce qui impose, pratiquement, une réalisation en logique cablée. Parmi les fonctions réalisées par le module vidéo sur certaines machines, on cite: la modification des couleurs des pixels, le fenêtrage, le zoom et la translation de l'image.

Notre travail porte sur la parallélisation du module graphique (partie qui génère les objets graphiques élémentaires). Pour valider cette parallélisation, on s'intéressera plus particulièrement aux algorithmes typiques qui sont le tracé de segment et le remplissage.

IV - L'ACCROISSEMENT DES PERFORMANCES:

Pour tenter d'abaisser, d'une manière sensible, les temps de réalisation des images de synthèse, on s'oriente vers l'idée de chercher de nouvelles architectures de machines. On distingue deux directions principales de recherche sur ces nouvelles architectures: les circuits spécialisés et le parallélisme.

A - LES CIRCUITS SPECIALISES:

Depuis quelques années, on trouve sur le marché des circuits spécialisés à trés grand taux d'intégration déstinés à l'infographie. Ces circuits, permettant de construire à un coût moindre des systèmes graphiques plus puissants, peuvent remplacer les classiques implémentations matérielles infographiques. Généralement, les circuits spécialisés sont utilisés soit comme des

modules spéçialisés, soit comme des modules graphiques. Quelques exemples d'architectures graphiques à base de circuits spécialisés: "Geometry Engine" et "Pixel-Planes".

Geometry Engine [CLA 80], [CLA 82]:

C'est un circuit spécialisé qui effectue quelques opérations géométriques courantes telle que le fenêtrage (Clipping) par un prisme de vision, les transformations affines définies à l'aide d'une matrice 4X4, et le changement d'echelle, simultanément avec la transformation perspective, dans le cas 3D. Il se compose de quatre unités identiques chacune possédant un octet pour l'exposant et vingt bits pour la mantisse. Il peut effectuer les opérations en virgule flottante, avec une structure simple de cinq éléments: une UAL, trois registres, et une pile.

Pixel-Planes [FUC 81]:

Ce circuit spécialisé se compose d'un pré-processeur, qui convertit les données polygonales dans une forme adéquate, et d'un système de cellules-mémoires à raison d'une par pixel. Il est prévu pour évaluer simultanément les valeurs d'une fonction linéaire $F(x,y)=ax+by+c$ pour tous les pixels (x,y) de la mémoire d'image. Chaque cellule-mémoire possède une UAL d'un bit, un bit d'inhibition qui sert de masque et une certaine quantité de mémoires stockant la valeur de la dernière fonction F(x,y) calculée, la couleur et la profondeur actuelles du pixel. Ceci permet d'exécuter rapidement les procédures graphiques les plus coûteuses: remplissage des polygones, élimination des parties cachées par l'algorithme du tampon en Z, lissage de type Gouraud en ramenant tous ces problèmes au calcul d'une certaine fonction linéaire F(x,y) en chaque pixel (x,y). Ces procédures sont effectuées polygone par polygone. Signalons, l'existence d'une extention de Pixel-Planes, appelée Pixel-Powers, pouvant évaluer des fonctions quadratiques de forme $axx+bxy+cyy+dx+ey+f$ simultanément pour chaque pixel (x,y).

B - LE PARALLELISME:

L' idée principale du parallélisme est de découper la tâche de la génération d'une image en de multiples sous-tâches, chaque sous-tâche étant traité par un processeur en même temps que les autres. A partir de ce principe, on distingue trois directions de recherche pour la parallélisation en synthèse d'images:

1 - Par partitionnement de l'espace objet (Partage objet):

On découpe la scène en sous-espaces, chacun étant traité par un processeur. Parmi les réalisations répondant à ce principe, on trouve:

Machine d'Ullner [ULL 83]:

C'est une machine pipeline. La scène est supposée constituée d'objets polyédriques, chaque polygone possédant son propre processeur. Comme l'algorithme de visualisation utilisé est celui du tracé de rayons, chaque processeur reçoit un rayon et calcule le point d'intersection, s'il existe, avec le polygone qui lui est associé.

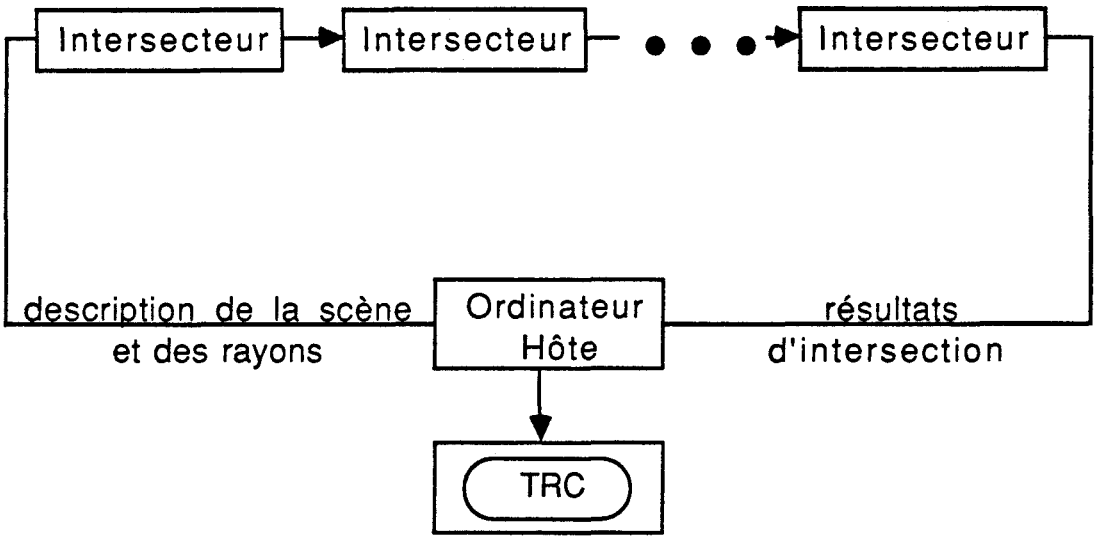


Fig 5

Machine de Weinberg:

Elle se compose de trois parties fonctionnelles:

- Un ensemble de processeurs objet qui, traitent chacun un trapèze (avec deux bords horizontaux).

- Un ensemble de comparateurs, montés en pipeline, pour construire une liste ordonnée de l'image à afficher.

- Un ensemble de filtres qui, vont calculer la couleur d'un pixel, à partir d'une sous-liste ordonnée en Z.

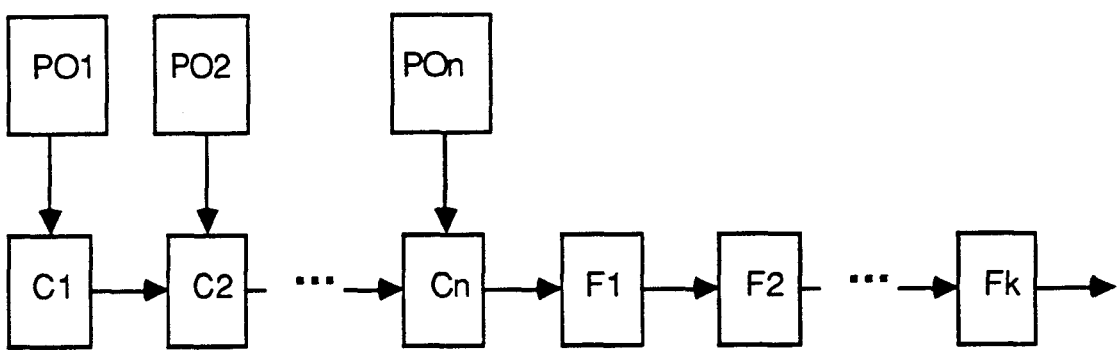


Fig 6

Machine de l'Université de Calgary [CLE 83]:

Elle est constituée d'un réseau matriciel de l'ordre de 10x10 processeurs. La scène est découpée en volumes rectangulaires élémentaires, associés à chacun des N processeurs. Chaque processeur stocke l'information concernant les surfaces qui intersectent le sous-espace dont il assure le contrôle. Comme l'algorithme de visualisation utilisé est le tracé de rayons, chaque rayon est transmis d'un processeur à un autre de la même manière que le rayon lumineux traverse la scène. Quand un processeur reçoit

le rayon, il teste si celui-ci intersecte les surfaces internes à son sous-espace.

Machine de Dippe et Swensen [DIP 84]:

Elle est assez proche de celle de l'Université de Calgary. La différence fondamentale provient du fait que le découpage de la scène est adoptif, de façon à uniformiser quelque peu la charge de travail de chaque processeur.

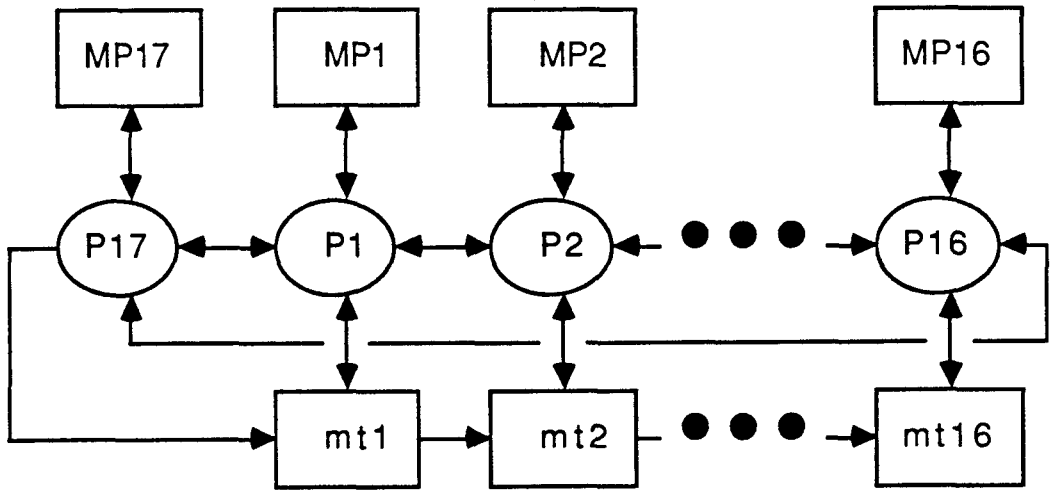
2 - Par partitionnement de l'espace image (Partage image):

L'idée est de décomposer l'écran en régions, chacune d'elles étant traitée par un processeur. Cette subdivision peut être composée de zônes comprenant des pixels dispersés, des zônes comportant plusieurs lignes de balayage consécutives, de zônes carrées, ... A la limite on pourrait envisager un processeur par pixel. Parmi les réalisations répondant à ce type de parallélisation, on trouve:

MAP (Multiprocesseur Associatif Parallèle) [COR 81]:

Cette machine, réalisé dans le laboratoire d'informatique de Lille, est constituée de seize processeurs, chacun d'eux s'occupe du traitement relatif à un seizième de l'image à afficher. En fait, c'est la mémoire de trame qui est divisée en seize régions, chacune d'elles formant une partition de l'écran.

Puisque les processeurs de traitement n'ont pas la possibilité d'adresser directement la mémoire de trame, c'est un dix-septième processeur qui assure le calcul des adresses et le contrôle des traitements.



MPi: Mémoire de travail du Processeur i
Pi: Processeur i
mti: Partie de la mémoire de trame calculée par Pi

Fig 7

Ce terminal a été doté de logiciels élémentaires, selon la terminologie de lucas, classiques au traitement (manipulation de la mémoire de trame), et, à la synthèse d'images (tracé de segments de droites, de cercles).

Le CAP (Cellular Array Processor) [HIR 85]:

Elle est constituée d'un tableau de 8x8 processeurs:

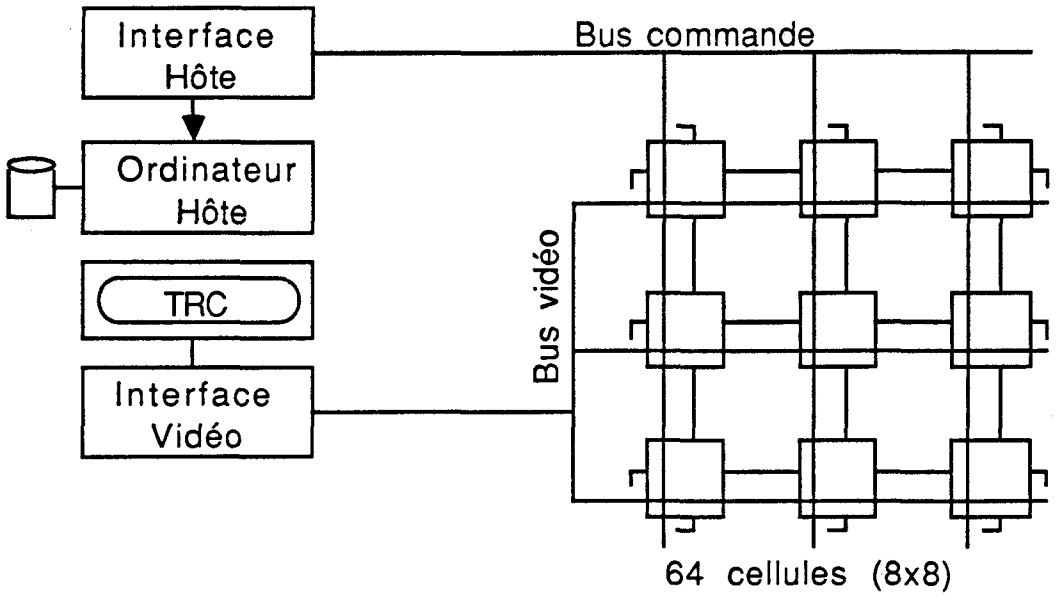


Fig 8

L'une des particularités de cette machine est qu'elle autorise différents découpages de l'espace image par programmation. Elle utilise le tracé de rayons pour la visualisation des images, et un arbre pour représenter la structure de données.

La machine CRISTAL (Calculateur Rapide d'Images par Suivi de Trajet Analytique de la Lumière) [BRU 86]:

Elle se présente sous forme d'un tableau de processeurs élémentaires (notés CPIX (Calculateur de PIXels)) et rassemblés par grappes. L'ensemble est contrôlé par un superviseur général (noté SUGE) qui assure la distribution et la synchronisation des tâches. La machine peut contenir jusqu'à 128 CPIX répartis dans 8 chassis. La grappe est un ensemble de ressources diverses et de processeurs reliés par un bus appelé bus local dont l'accés est géré par un contrôleur d'accés local (noté CAL). Un bus, appelé bus global, relie

les grappes entre elles. Ce dernier est géré par un contrôleur d'accés global (noté CAG). Un tampon FIFO, alimenté par le SUGE, stocke les séquences de tâches à exécuter.

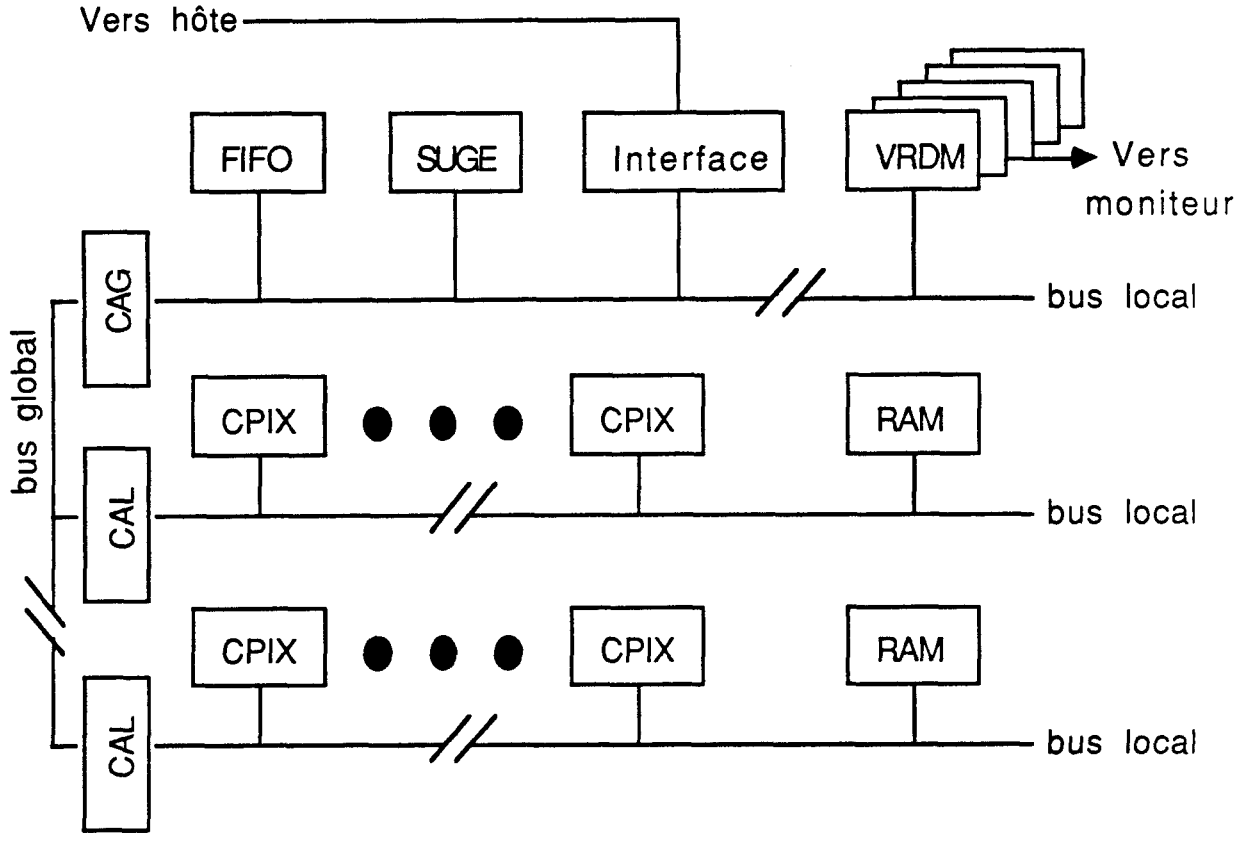


Fig 9

Le SUGE assure l'interface avec l(ordinateur hôte. Il s'occupe aussi de la détermination des pixels des zônes de l'écran qui sont réellement à calculer et les distribue entre les CPIX présents et disponibles. Chaque Cpix réalise le calcul complet de la couleur des pixels qui lui sont assignés. En général, la structure de données décrivant la scène est stockée dans la mémoire locale du CPIX.

3 - Par découpage du processus opératoire (Pipeline):

Chaque processeur s'occupe d'une opération bien détérminée (tri, calcul d'ombrage, ...). Parmi les réalisations répondant à ce principe, on trouve:

Geometry Engine [CLA 82]:

Elle comporte douze circuits identiques. Ces circuits sont disposés en pipeline.

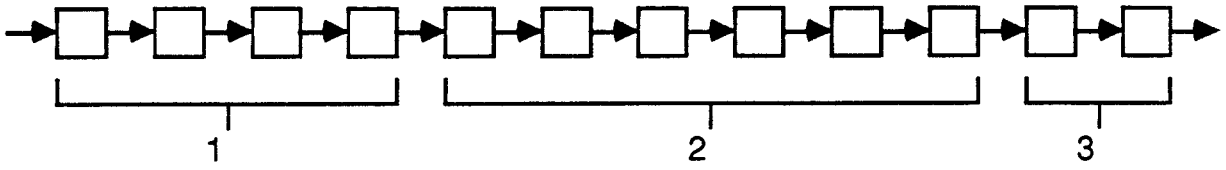


Fig 10

Les trois groupes du pipeline sont:

- Le groupe 1 s'occupe des transformations géométriques en deux où trois dimensions.

- Le groupe 2 se charge du découpage de la scène suivant une fenêtre (2 dimensions) où une pyramide tronquée (3 dimensions).

- Le groupe 3 effectue les opérations de projections perspectives où orthogonales ainsi que la mise à l'échelle (mise en écran).

Machine Périphérique d'Ordinateur d'Ullner [ULL 83]:

Elle se compose de trois étages:

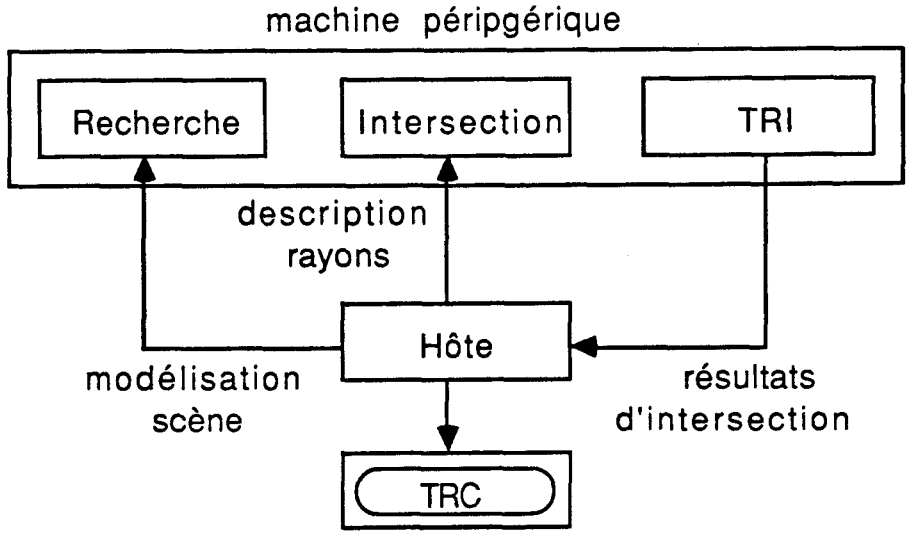


Fig 11

Le premier étage recherche les polygones dans la mémoire de description de la scène à modéliser. Le deuxième étage effectue le calcul des intersections. Le troisième étage réalise le tri des points d'intersections. Une fois que la traitement des polygones de la scène est terminé, le résultat final se trouve au niveau du troisième étage du périphérique.

Le LINKS-1 (de Tokyo Links Corporation) [NIS 83]:

Il est constitué de 64 processeurs. La machine possède un processeur maître RC et des calculateurs NCi. Les calculateurs NCi sont interconnectés de telle sorte qu'ils puissent être configurés sous forme de pipeline et sous contrôle du processeur maître.

V - CONCLUSION:

Pour optimiser les performances d'un système infographique, deux directions sont possibles:

* Les circuits spécialisés

* Le parallélisme

La première a donné naissance à certains processeurs graphiques spécialisés. La deuxième a permis certaines réalisations. Notre étude porte sur la fusion de ces deux directions, c'est àdire sur la parallélisation des processeurs graphiques spécialisés. Et pour celà, on commencera par étudier les processeurs graphiques spécialisés existants afin d'en choisir le plus complet, et seulement après, on s'intéressera à la parallélisation de ce dernier. Pour valider la parallélisation, on étudiera plus particulièrement les algorithmes les plus typiques de la synthèse d'images: le tracé de segment et le remplissage. Donc, le deuxième chapitre portera sur l'étude des différents processeurs graphiques spécialisés existants et le troisième sur celle de la parallélisation. Le quatrième et le cinquième chapitre sont consacrés respectivement aux deux solutions retenues.

CHAPITRE II

ETUDE DES DIFFERENTES POSSIBILITES DES PROCESSEURS GRAPHIQUES SPECIALISES

SOMMAIRE

I-INTRODUCTION:

A ses débuts, la synthèse d'images produisait des dessins au trait c'est à dire des graphiques à base de lignes, mais, très vite, la nécessité d'obtenir des images figuratives ou plus réalistes a exigé des traitements plus évolués et des techniques plus complexes. L'accroissement des temps de calcul, qui en a résulté, pénalise l'interactivité et la production graphique en empêchant les traitements en temps réel.

Pour pallier cà, il faudrait obtenir de hautes performances en vitesse d'execution, capacité de stockage et de traitement, et en temps de réponse et d'affichage. Mais toute recherche de performance peut aujourd'hui se décomposer en deux parties:

* La réalisation de composants de hautes performances

* La simultanéité obtenu par la mise en parallèle de plusieurs de ces composants

Si, pour le graphique, la première partie relève en particulier des constructeurs et conduit à des solutions figées, la seconde apparait beaucoup plus souple et ouverte.

Ceci a donné naissance à des processeurs graphiques spéçialisés. Parmi ces derniers on trouve:

* Le 82786 de chez INTEL

* Le TMS 34010 de chez TEXAS INSTRUMENTS

* Le AM95C60 de chez ADVANCED MICRO DEVICES

* Le RGP de chez NATIONAL SEMICONDUCTOR CORP.

* Le AGDC de chez NEC ELECTRONICS

Ce chapitre consiste en l'étude des différentes possibilités de ces processeurs graphiques spéçialisés afin de pouvoir établir une comparaison entre eux.

II - PROCESSEURS GRAPHIQUES SPECIALISES:

1 - INTEL 82786 [ANI 87], [INT 87], [MMS 87]:

L'Intel 82786 est formé de deux processeurs indépendants: "GP" ("Graphics Processor") processeur graphique et "DP" ("Display Processor") processeur d'affichage, montés sur un seul circuit VLSI et qui manipulent respectivement graphique, texte, et, affichage, fenêtres.

Le 82786 contient, comme tout processeur graphique, au moins une interface vidéo ("display": affichage), une mémoire graphique et une interface hôte.

Grâce au fait que le "GP" et le "DP" manipulent indépendemment et respectivement le graphique et l'affichage, le 82786 peut tracer 2.5 million pixels/sec et 25 mille caractères/sec ceçi en utilisant une horloge vidéo de 25 MHz et une DRam conventionnelle.

Le CPU contrôle les ressources du coprocesseur graphique "GP" à travers un bloc de 128 bytes de registres de contrôle interne.

Le controlleur permet le rafraîchissement et l'accés à la DRam. La mémoire est adressable sur 22 bits. Cet espace d'adressage est divisé en espace d'adressage de mémoire graphique et en espace d'adressage de mémoire système. La taille de la mémoire graphique peut atteindre 4 Mbytes. La mémoire graphique contient les données d'affichage, les caractères, et les instructions graphiques.

Pour contrôler les accès à la mémoire graphique, la "BIU" maintient une priorité programmable pour tout demandeur de contrôle de bus.

Le 82786 supporte un nombre variable de bits/pixel et autorise l'utilisation de combinaisons variées dans différents bit-maps. Il peut fonctionner en deux modes: mode maître ou mode esclave.

Les instructions graphiques du 82786 sont construites sous un format simple: 8 bits pour le code de l'opération ("opcode"), le bit GECL flag ("graphics-end-of command list"): bit d'indication de fin de liste de commandes, et sa liste de paramètres associée.

Pour accélèrer les opérations graphiques, le 82786 coopère avec le microprocesseur hôte. Puisque le 82786 n'est pas un processeur microprogrammable, il n'a pas besoin de fonctionner dans son propre environnement et peut travailler dans un environnement existant. A titre d'exemple l'IBM PC/AT peut servir comme hôte pour le 82786.

En multitâches, le 82786 peut supporter les demandes d'execution de plusieurs tâches concurrentes, et le "GP" peut modifier les bits-maps de différentes tâches pendant que le "DP" trace seulement ceux qui intéressent le plus l'utilisateur.

Le 82786 peut fonctionner en mode accéléré:

* Horloge vidéo externe de 50 MHz: résolution de 4 bits/pixel

* Horloge vidéo externe de 100 MHz: résolution de 2 bits/pixel

* Horloge vidéo externe de 200 MHz: résolution de 1 bit/pixel

Une très rapide palette vidéo Ram peut être utilisée pour restaurer une haute résolution de couleur d'une information stockée en basse résolution.

Enfin pour une très grande résolution graphique, on peut utiliser plusieurs 82786.

2 - TMS 34010 [CAR 86], [MMS 87], [TEX 87]:

Un vrai processeur graphique ne doit pas dépendre du hôte pour n'importe quelle opération graphique et doit avoir son propre compilateur de haut niveau.

Le processeur graphique TEXAS INSTRUMENTS TMS 34010, processeur à 32 bits, optimise les opérations graphiques du système en fonctionnant comme un processeur général. Sa programmation générale, augmentée par l'implémentation d'opérations graphiques spéçiales en hardware, lui donne l'avantage par rapport aux contrôlleurs graphiques et aux microprocesseurs généraux.

Le 34010 peut prendre en charge des opérations sans le processeur hôte. Ceci facilite sa programmation en langage de haut niveau.

Cependant, le processeur hôte est libre pour prendre en charge d'autres fonctions importantes du système comme la gestion de mémoire et de file, entrées/sorties et les différents périphériques, le contrôle du clavier ... etc

Le 34010 interface directement avec "DRams" ("Dynamic Rams") et "VRams" ("Vidéo Rams") et supporte le décodage d'instruction et la manipulation des données à 6 Mbytes/sec. Il gère aussi les signaux de contrôle du "CRT" qu'utilisent la plus part des systèmes d'affichage.

D'autres processeurs peuvent communiquer avec le 34010 à une vitesse qui au maximum ne dépasse pas 5 Mbytes/sec. L'interface hôte est configurable pour 8 ou 16 bits (bus) et réclame un accés drecte bidirectionnel pipeliné ("DMA": Direct memory access) entre la mémoire locale et les registres de l'interface hôte. L'interface hôte

peut communiquer avec d'autres 34010. Ceci est important dans la partage des tâches, sur plusieurs processeurs graphiques, qui est une technique qui augmente beaucoup la performance.

Le 34010 supporte les plus importantes structures de données utilisées en graphique: bits, bytes, pixels, champs ("fields"), tableaux à 2 dimensions ("2D-arrays"). Il existe 2 modes d'adressage pour les données graphiques: linéaire ("linear") et cartésien X-Y.

Le 34010 a 2 jeu d'instructions: un ensemble d'instructions primitives ("RISC": Reduced Instruction Set Computer) qui contient toutes les opérations, à un cycle, d'un microprocesseur général, comme les sauts et les appels ("Jumps and Calls"), les décalages et rotations ("Shifts and Rotates"), les opérations arithmètiques sur les booléens et les entiers ("Integer and Boolean Arithmetic"), les transferts ("Moves"),... etc, et un ensemble d'istructions complexes ("CISC": Complex Instruction Set Computer") qui contient les opérations graphiques fondamentales et importantes comme le transfert de bloc de pixels ("Pixblts": Pixel Block Transfers), le tracé de ligne ("Line"), le remplissage ("Fill"),... etc.

Le jeu d'instructions du processeur graphique incorpore plusieurs mécanismes pour la manipulation des pixels ou des tableaux de pixels. L'opération Pixblts du 34010 exige que le matériel supporte les fenêtres ("Windowing"), le maskage des plans ("Plane Masking"), la transparence ("Transparency"), les opérations d'expansion ("Binary-to-Color expand opération").

Le 34010 supporte 16 opérateurs logiques et 6 arithmétiques pour le traitement des pixels couleur. Pour augmenter l'efficacité de la programmation, le 34010 a 2 files de registres, chacune contient 15 registres, de 32 bits, généraux (utilisés pour les adresses et pour les données), un pointeur de pile SP ("Stack Pointer"). Au total le processeur graphique a 31 registres programmables utilisables. Grâce à ces registres les algorithmes graphiques deviennent plus efficaces puisque on réduit le nombre de données à transférer.

La performance croît en introduisant dans le système plus de traitements parallèles. Le 34010 a un haut niveau de parallélisme interne avec le cache d'instructions (mémoire), son registre Barrel Shifter, "Alu" unité arithmétique et logique ("Arithmetic Logic Unit") et son interface mémoire locale.

Les 256 bytes du cache d'instructions supporte simultanément l'accés à la mémoire et aux registres. Le cache réduit le nombre de recherche d'instructions que le processeur graphique doit faire. Ceci accélère les algorithmes graphiques et surtout ceux qui contiennent plusieurs boucles d'unstructions itératives.

Le registre Barrel Shifter peut faire la rotation de n'importe quel champ, en 32 bits, en un seul, 160 ns, cycle opération. Le Barrel Shifter accélère les opérations de manipulations de bits de champs utilisées en graphique.

L'Alu (32 bits) est capable de traiter plusieurs pixels en parallèle. Quand la taille du pixel est connue, l'Alu se configure comme une série de petites Alu, qui traitent simultanément des opérations booleènnes et arithmétiques sur les pixels.

Le pipeline augmente la performance du système. Pendant un seul cycle, le 34010 peut lire de 2 registres, écrire dans un registre, décoder une instruction, executer l'instruction courante et commencer ou compléter un cycle de mémoire locale. Ceci le rend un processeur général solide et lui donne la possibilité d'executer les instructions à une vitesse de 6 MIPS (Million d'Instructions Par Seconde) quand son horloge est de 50 MHz et execute hors du cache.

La tolérance de l'utilisateur pour ce qui est des applications graphiques est que le temps de réponse de l'écran tourne autour d'un tiers de seconde. Le 34010, avec 50 million bits / sec ("bps") pour l'opération "fill" et 25 million bits / sec pour l'opération "pixblts" permet l'apparition presque instantannée de la plus part des images.

Les outils de développement de logiciels opère sur un IBM PC, PC/XT, PC/AT et compatibles ou sur DEC VAX système. Le minimum d'outils dont on a besoin pour développer des logiciels est: le "bagage" Assembleur, le compilateur C, et une carte de développement de logiciels qui contient un debugger.

Le "bagage" Assembleur consiste en un "Linker" (éditeur de liens), un "Archiver" (archiveur), "Rom Utilities" (utilitaires Rom), et un simulateur pour les versions PC.

La carte de développement de logiciels permet aux programmeurs de modifier directement le contenu des registres du 34010.

Un logiciel écrit en langage C peut appeler des routines écrites en langage Assembleur 34010.

3 - AM95C60 [AMD 86], [AMD 87], [MMS 87]:

Le AM95C60 QPDM ("Quad Pixel Dataflow Manager") est un CMOS processeur graphique qui conduit 4 tableaux bit-map de mémoire. Fonctionnant à une vitesse d'horloge maximale de 20 MHz, il peut interfacer à 8 ou 16 bits de bus.

Il peut tracer des vecteurs à 3.3 million pixel/sec où placer du texte à 45 mille caractères/sec. Cette performance donne la possibilité aux utilisateurs de mélanger texte et graphique dans le bit-map. Le AM95C60 contient des primitives graphiques et interface avec GKS, NAPLPS, VDI ("Virtual Device Interface") et les logiciels standards ("Software Standards").

Le processeur est totalement cascadable et peut gérer 256 "Memory Planes" sans que celà affecte sa performance système.

D'autres possibilités comme le fenêtrage ("Windowing"), les facteurs de zoom indépendants de X et Y ("Independant X and Y Zoom Factors"), "Pan and Scroll", "Picking", "Clipping" et "Logical pen size".

Le AM95C60 supporte aussi le tracé de vecteurs, cercles, et arcs anti-aliassés en utilisant des styles de lignes très variants définis par l'utilisateur.

4 - RGP [MMS 87]:

Le RGP ("Raster Graphics Processor") est un CMOS, 20 MHz, microprocesseur totalement programmable destiné au graphique. Il contient la logique de l'état de la machine pour maximiser la performance des primitives graphiques.

En plus de son jeu d'instructions générales, le RGP contient des instructions primitives pour la génération de modèles de lignes et poly-lignes, "BitBlts" ("Bits Block Transfer"), polygones pleins ("Filled Polygons") et supporte les caractères ("Bit-mapped Fonts").

Le RGP peut atteindre une vitesse extrêmement haute: 10 million pixel/sec pour modèles de lignes, 160 million pixel/sec pour "Fill", 80 million pixel/sec pour "BitBlts" et 100 mille caractères/sec pour "Bits mapped Fonts". Cette performance est indépendante de la profondeur de la couleur, ceçi est dû à l'architecture plane du pixel.

Toutes les opérations de traçage sont accomplies en coordonnées X - Y: le RGP convertit automatiquement en adresses linéaires quand il accède au buffer.

Le RGP contient aussi le hardware du "Clipping", qui opère sans affecter la performance, et supporte aussi le rafraîchissement de l'écran.

5 - AGDC [MMS 87]:

Le AGDC ("Advanced Graphics Display Controller") est un seul, VLSI, coprocesseur graphique à haute performance. Sa haute vitesse de fonction de traçage graphique inclut le coloriage ("Painting"), les "BitBlts" opérations, les commandes de demande ("Inquiring commands"), les opérations logiques avec et/ou entre les plans, le "Clipping" et "Picking".

La vitesse de traçage est de 500 nsec/pixel à 8 MHz, 400 nsec/pixel à 10 MHz, 32 million bits/sec pour "BitBlts". La haute performance du système est accomplie à travers 3 stades de traitement pipeliné de commandes du CPU au traçage du processeur en utilisant un préprocesseur.

Le AGDC peut aussi accèder simultanément aux pixels du buffer graphique quand ils sont sous les configurations mode pixel et mode plan.

6 - CONCLUSION:

Après comparaison de ces processeurs spécialisés, qui présentent, en général, la même structure, on a opté pour le TMS 34010. En effet, ce dernier comporte un cache mémoire d'instructions, un registre barrel shifter, et offre plus d'opérateurs logiques et arithmétiques sans oublier la reconfiguration en parallèle de son unité arithmétique, ... etc

III - LES ARCHITECTURES MATERIELLES [FAK 83], [FLY 72]:

A - LA STRUCTURE LOGIQUE:

Elle détermine l' organisation des processeurs et les relation entre eux. On distingue deux organisations principales:

1 - L'ORGANISATION HIERARCHIQUE (OU VERTICALE):

Dans cette organisation, il y a un processeur maître et plusieurs esclaves. Tous les processeurs sont logiquement différents. Les communications interprocesseurs transitent par le maître. Chaque processeur peut devenir maître.

2 - L'ORGANISATION NON HIERARCHIQUE (OU HORIZONTALE):

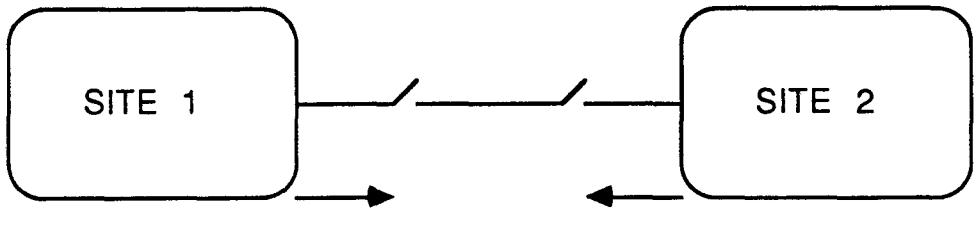
Dans cette organisation, tous les processeurs sont logiquement équivalents. Chaque processeur peut communiquer avec n'importe quel autre processeur. Le contrôle et la coordination peuvent être ou non assurés par un processeur maître.

B - LA STRUCTURE PHYSIQUE:

Elle détermine la manière du cheminement de l'information dans le système, la configuration de l'outil de communication et la topologie d'interconnexion.

1 - L'OUTIL DE COMMUNICATION:

L'information transite entre les processeurs soit par l'intermédiaire d'une mémoire commune (commutation de ligne (voir Fig 1)) où d'un bus (commutation de ligne (voir Fig 2)).



SIGNAUX SYNCHRONES

Fig 1

Dans ce cas, tous les transferts se font par l'intermédiaire d'une mémoire commune, et les processeurs n'ont pas d'accés direct les uns vers les autres.

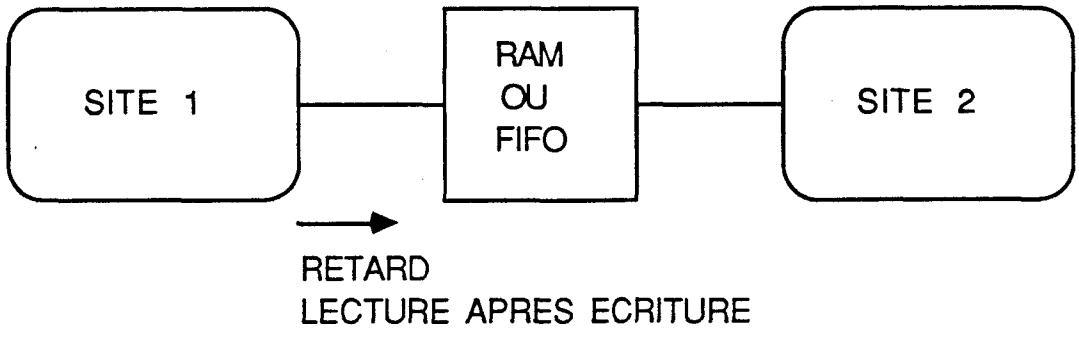
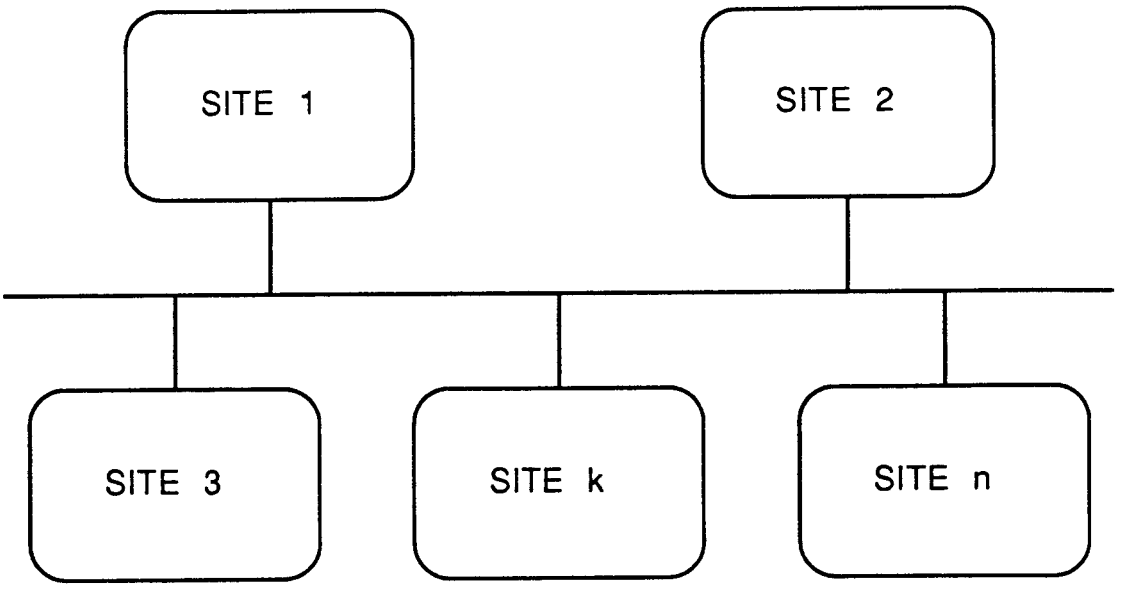


Fig 2

Inversement, dans ce cas, tous les processeurs communiquent entre eux par l'intermédiaire du bus.

2 - LA TOPOLOGIE D'INTERCONNEXION:

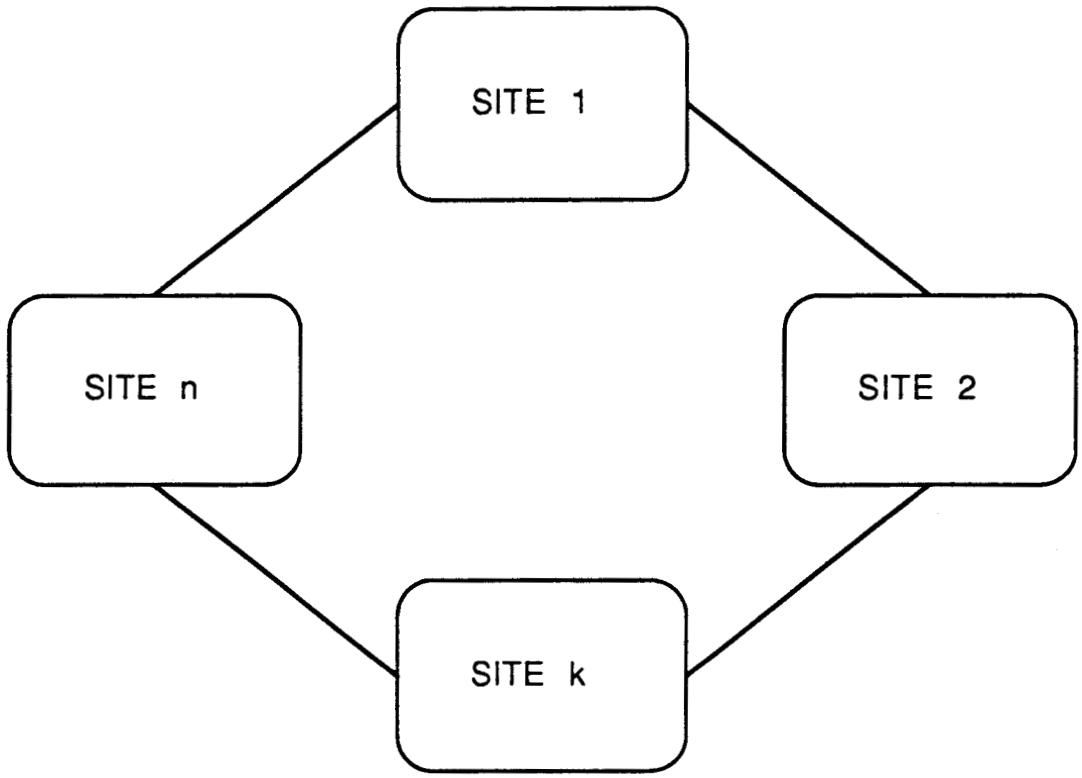
Cà consiste, à considérer comment sont connectés ensemble les différents éléments, ceci étant très important pour caractériser la fiabilité ainsi que les possibilités d'extension du système. On distingue quatre schémas d'interconnexion de bases :



Partage d'un outil commun d'interconnexion

Fig 3

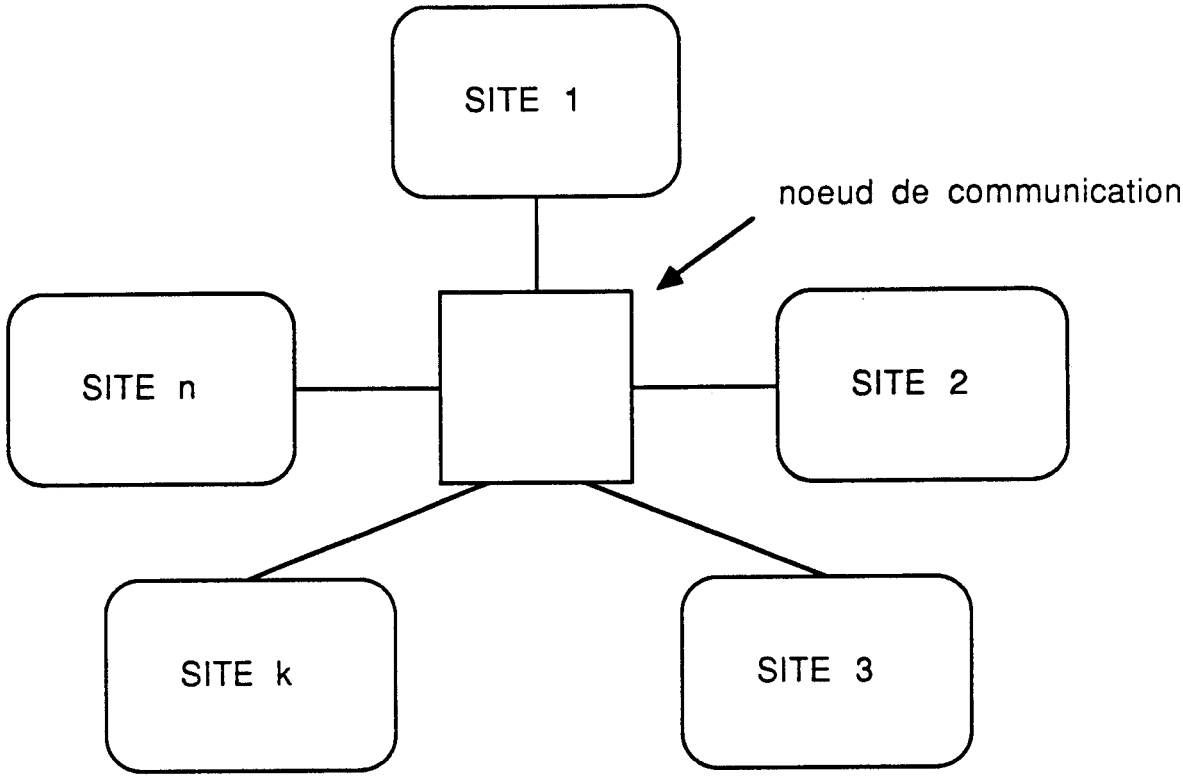
Dans ce cas, il y a partage d'un outil commun de communication. Ce qui veut dire, à un instant donné, il n'y a que deux sites qui peuvent communiquer. Dans cette configuration, chaque site peut communiquer directement avec n'importe quel autre site.



Interconnexion en anneau

Fig 4

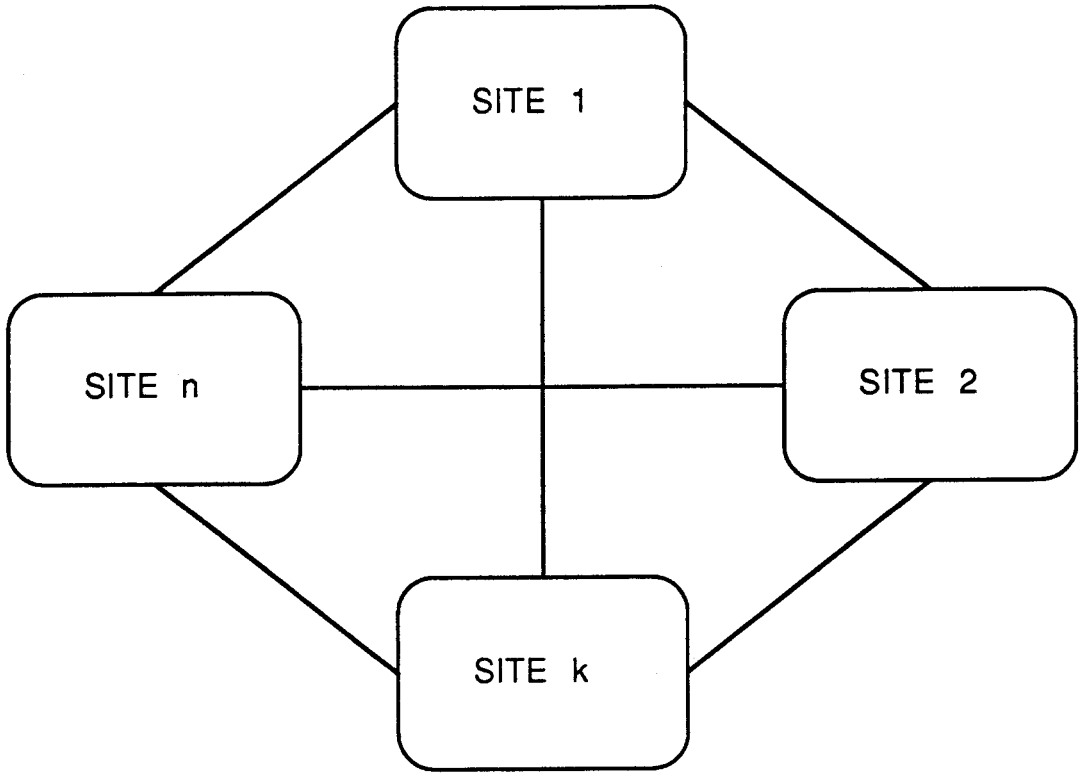
Dans cette configuration, chaque site ne peut communiquer directement qu'avec deux sites qui sont respectivement les voisins de droite et de gauche.



Interconnexion en étoile

Fig 5

Dans ce cas, chaque site peut communiquer directement avec n'importe quel autre site par l'intermédiaire du noeud de communication.



Interconnexion complète

Fig 6

Dans cette configuration, chaque site peut communiquer directement avec n'importe quel autre site.

C - LE MODE D'INTERACTION:

Ce critère détermine le facteur de couplage, la nature et le taux de communication entre les sites. Par conséquent, il détermine le taux d'occupation des ressources partagées.

Il y a trois sortes de couplages: le couplage fort, modéré, et faible.

1 - LE COUPLAGE FORT:

Les ressources nécessaires au traitement sont partagées par les sites. Dans ce cas on a:

* Partage d'une mémoire commune. Et généralement, chaque processeur a une mémoire locale et une interface de communication.

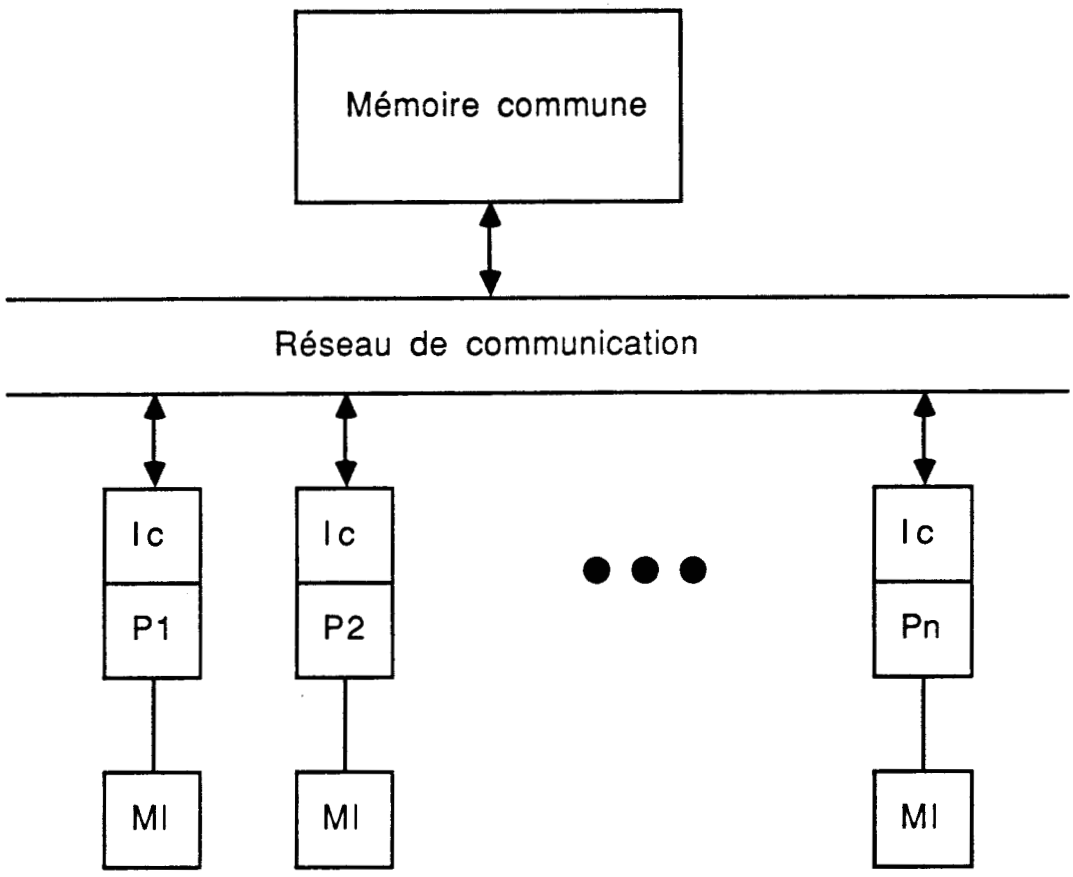


Fig 10

Ml : mémoire locale
Ic : interface de communication

* Un système d'exploitation commun permettant la coordination des sites.

* Les sites sont généralement homogènes. Cependant ils permettent la banalisation des fonctions de programmation et l'égalité des puissances de traitement.

Ce type de couplage nécessite des moyens de synchronisation pour le partage des ressources communes.

2 - LE COUPLAGE FAIBLE:

Chaque site est pourvu de moyens matériels et logiciels lui permettant une indépendance de traitement. On parle alors de réseaux d'ordinateurs.

3 - LE COUPLAGE MODERE:

C'est le couplage intermédiaire qui correspond à une combinaison des couplages faible et fort, qui représentent les cas extrêmes du couplage.

D - LE MODE DE FONCTIONNEMENT:

Il y en a quatre:

1 - LE MODE "SISD":

("Single Instruction Stream, Single Data Stream") Un flux d'instruction opère sur un flux de donnée:

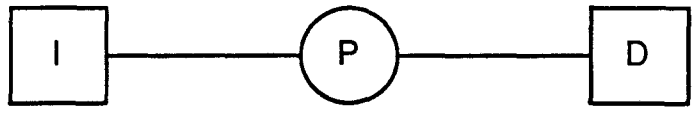


Fig 11

2 - LE MODE "SIMD":

("Single Instruction Stream, Multiple Data Stream") Un flot d'instructions opère sur plusieurs flots de données:

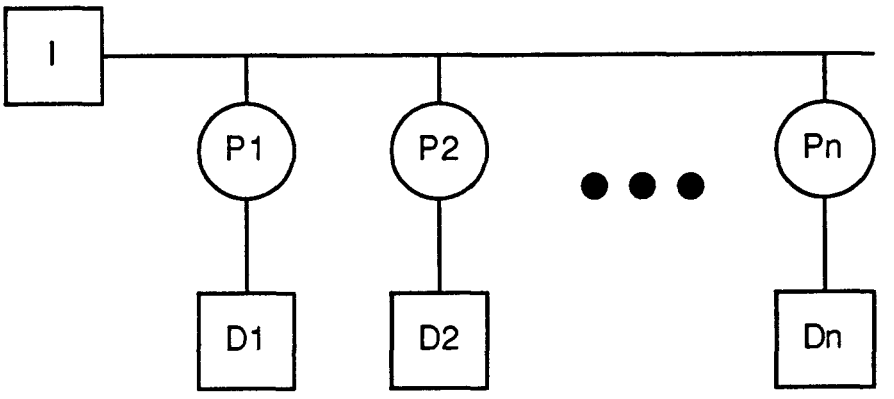


Fig 12

3 - LE MODE "MIMD":

("Multiple Instruction Stream, Multiple Data Stream") Plusieurs flots d'instructions opèrent sur plusieurs flots de données:

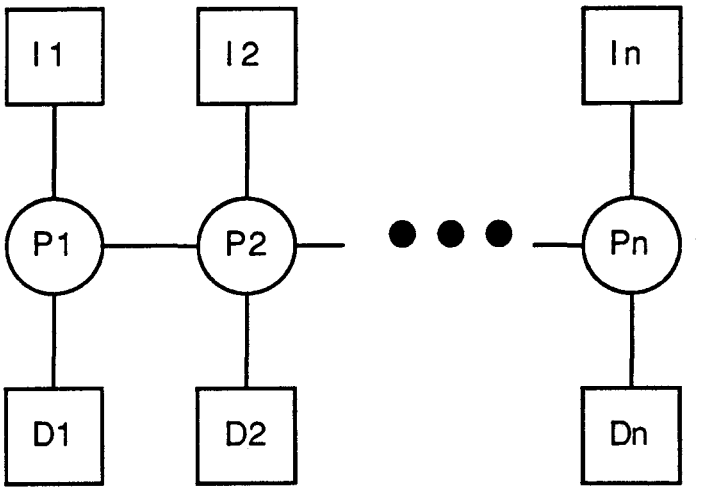


Fig 13

4 - LE MODE "MISD":

("Multiple Instruction Stream, Single Data Stream") Plusieurs flots d'instructions opèrent sur un flot de données:

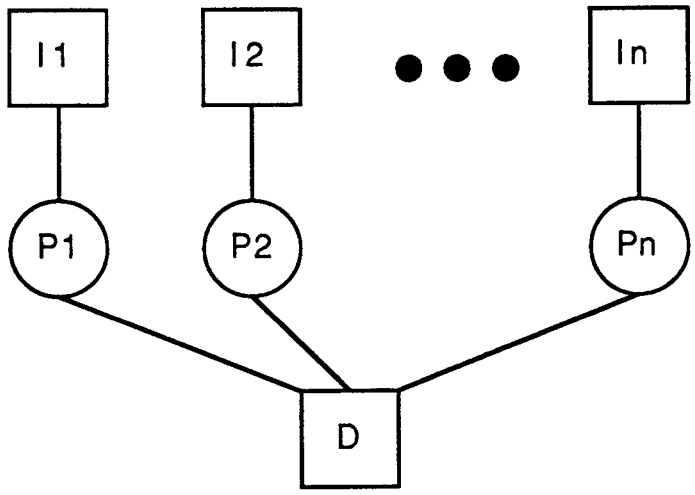


Fig 14

V- CONCLUSION:

Actuellement la tendance est à intégrer le maximum de fonctions au niveau du matériel, ceci étant dû au faible coût et à la rapidité des composants éléctroniques. Ce qui a donné naissance aux processeurs graphiques spéçialisés.

L'accroissement des performances d'un système infographique, passe par la parallélisation. Donc, pour optimiser les performances des processeurs graphiques spécialisés, il faudrait étudier leur parallélisation.

CHAPITRE III

ETUDE DE LA PARALLELISATION DES PROCESSEURS GRAPHIQUES SPECIALISES

SOMMAIRE

I - INTRODUCTION:

L'étude d'architectures adoptées passe par la combinaison des avantages des deux approches: le parallélisme et les processeurs graphiques spécialisés.

Le problème important concerne le stockage de l'image et le partage des accés à la mémoire correspondante.

Les approches du parallélisme sont le pipeline et le partage.

II - LE PIPELINE:

Le pipeline consiste en un groupement de processeurs, chaque processeur s'occupe d'une tâche donnée, par exemple:

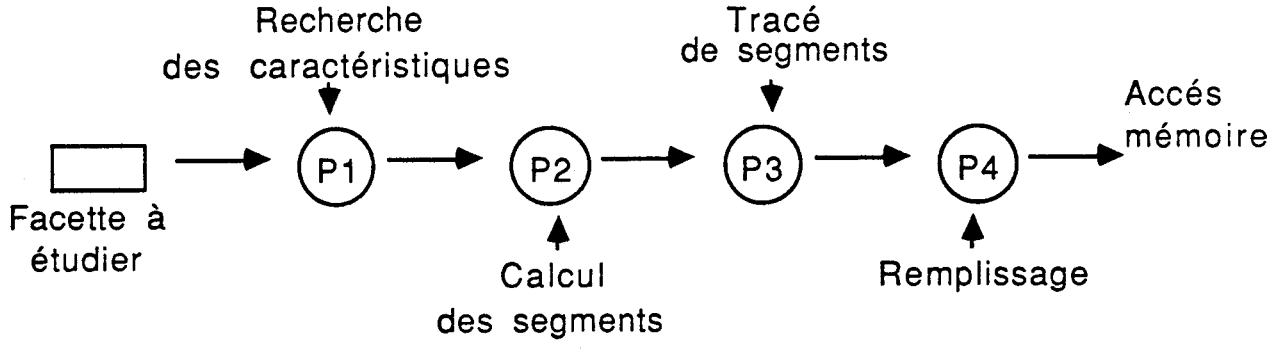


Fig 1

Le processeur spécialisé offre déjà toutes ces opérations. Donc, au lieu de grouper ces processeurs en un pipeline, il est plus avantageux de les utiliser en parallèle afin de profiter du maximum de leurs puissances.

III - LE PARTAGE:

L'approche partage peut être réalisée de deux façons: le partage géométrique (ou exclusif) et le partage par objet (ou inclusif).

A - LE PARTAGE GEOMETRIQUE:

Le partage géométrique (où encore le partage de zône) revient à attribuer à chaque processeur sa propre partie de la mémoire d'image.

Exemple pour quatre processeurs:

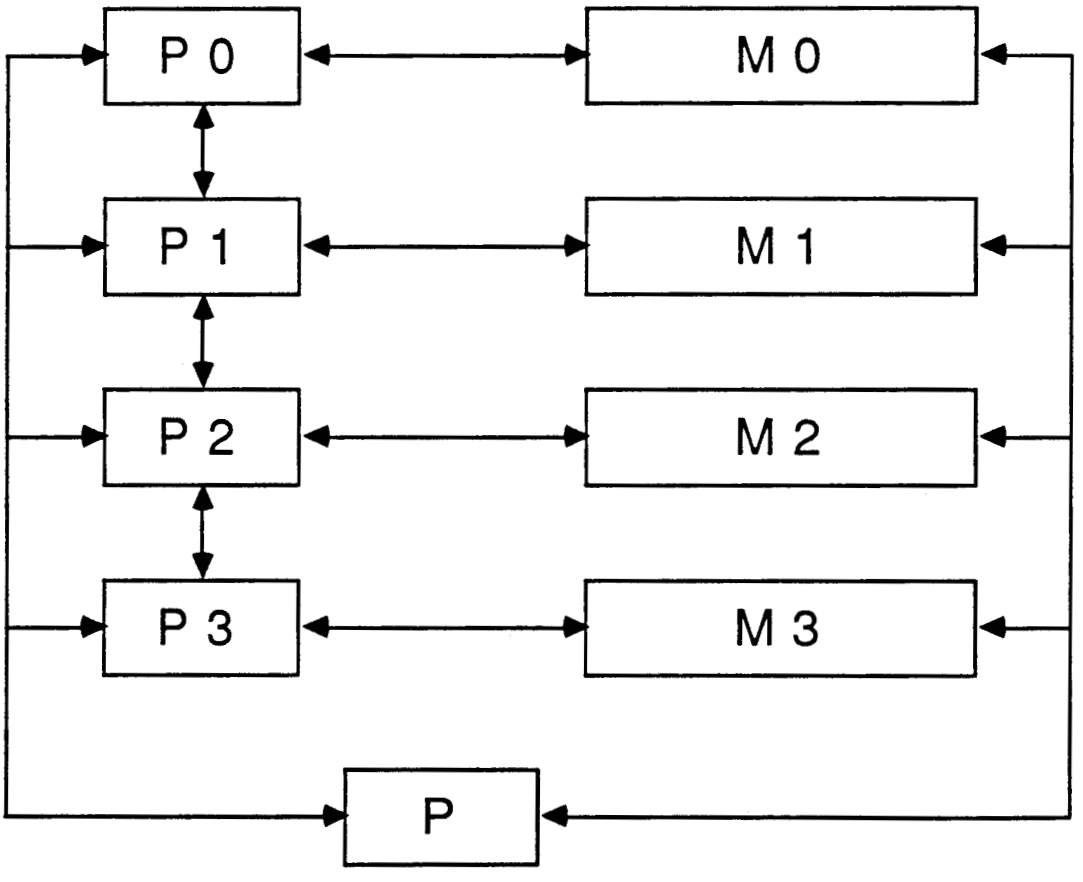


Fig 2

La mémoire image est la représentation matricielle de l'écran sous forme de mots mémoires où chaque mot représente la valeur numérique ou couleur à affecter au pixel (voir Fig 3).

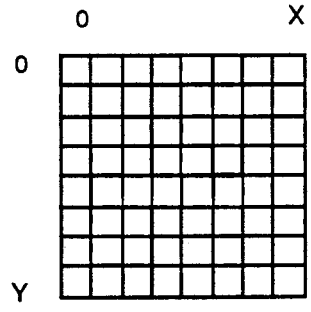


Fig 3

Les pixels sont accessibles à l'aide de couples de coordonnées (y,x).

Le partage exclusif peut être représenté par la manière suivante:

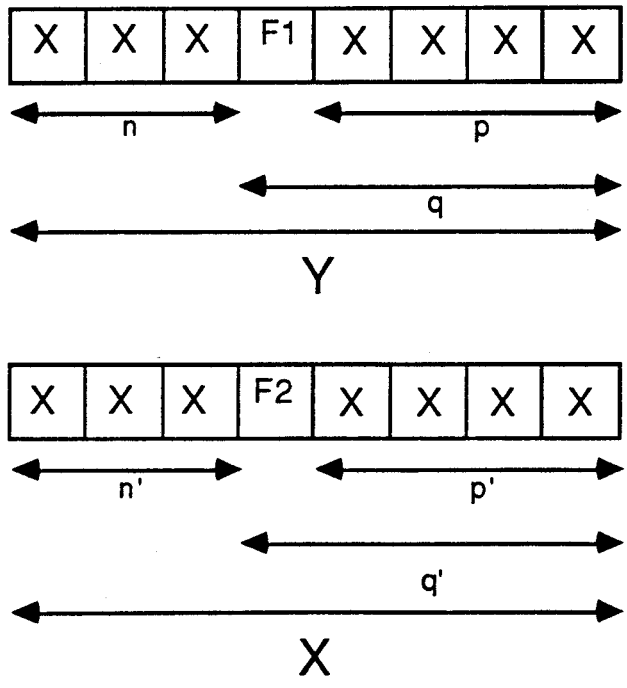


Fig 4

F1F2: identifie la fenêtre et par conséquent le processeur qui gère cette dernière.
pp': identifie taille de la fenêtre (= (2 puissance p)*(2 puissance p')).
qq': identifie taille du pavé (= (2 puissance q)*(2 puissance q')).
nn': identifie nombre de pavés constituant la mémoire image (= (2 puissance n)*(2 puissance n')).

Définitions:

Fenêtre : c'est une partie de l'écran qui est visible par un processeur.
Pavé : c'est une partie contigue de l'écran.

1- PREMIER CAS:

Cas où le pavé est de taille un, c'est à dire c'est le cas où chaque processeur s'occupe d'une partie contigue de l'écran.

Exemple pour 4 processeurs:

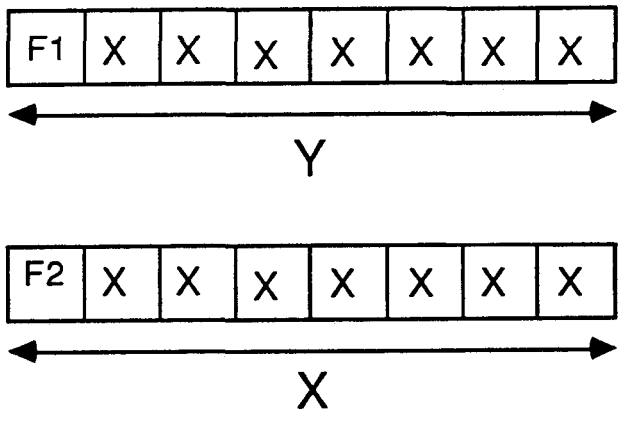


Fig 5

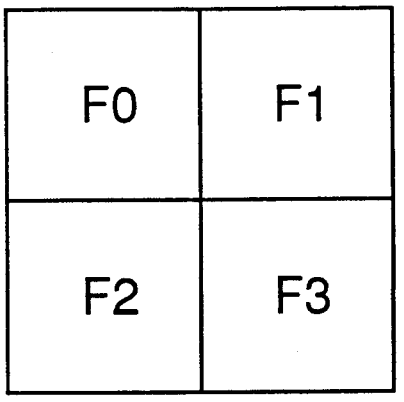


Fig 6

Soit X, Y la taille de l'écran en abscisse et ordonnée respectivement.

* F0: Fenêtre du processeur P0 qui correspond à:
  (0xxxxxxx, 0xxxxxxx)<=> ((0... Y/2 - 1), (0... X/2 -1))

* F1: Fenêtre du processeur P1 qui correspond à:
  (0xxxxxxx, 1xxxxxxx)<=> ((0... Y/2 - 1), (X/2... X-1))

* F2: Fenêtre du processeur P2 qui correspond à:
  (1xxxxxxx, 0xxxxxxx)<=> ((Y/2... Y-1), (0... X/2 -1))

* F3: Fenêtre du processeur P3 qui correspond à:
  (1xxxxxxx, 1xxxxxxx)<=> ((Y/2... Y-1), (X/2... X-1))

Cette configuration devrait bien marcher pour ce qui est du traçage de ligne, cercle, rectangle et du remplissage s'ils sont centrés sur le centre de l'écran. Puisque ces figures de base auraient comme axes de symétrie les axes de l'écran, et de ce fait en coupant la figure par ces axes, on obtiendrait 4 parties de figure semblables. Ce qui veut dire, que les 4 processeurs auraient à tracer les 4 parties égales de la figure en même temps. Donc au lieu de la tâche on aurait 4 sous-tâches s'executant en parallèle et dont le temps d'execution serait d'environ le quart du temps d'execution de la tâche.

Pour les tracés et les remplissages non centrés sur le centre de l'écran, on essayera de les dessiner au centre de l'écran (pour se ramener au cas précédent), et seulement après on les translatera pour les placer à la position souhaitée.

Essayons de traiter un exemple: celui du tracé de cercle.

Première étape:

Tracé du cercle au centre de l'écran.

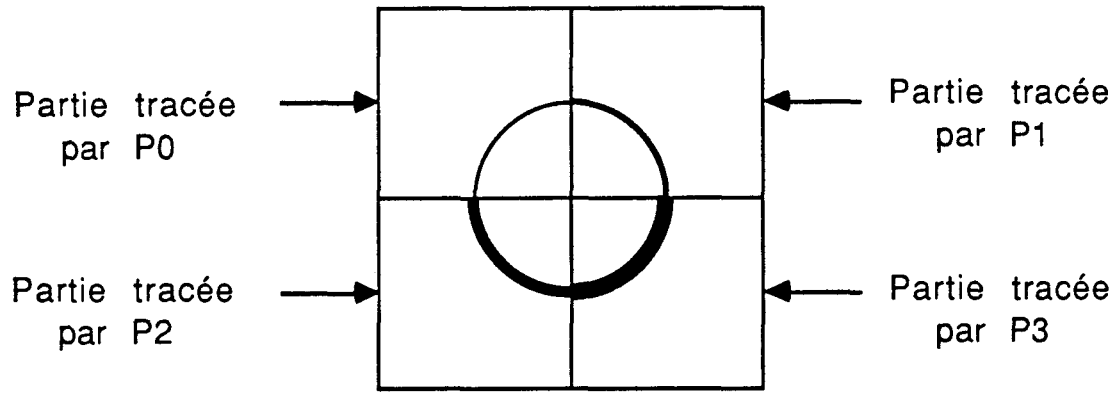


Fig 7

Le tracé des quatre parties se fait en parallèle.

Deusième étape:

Translation du tracé.

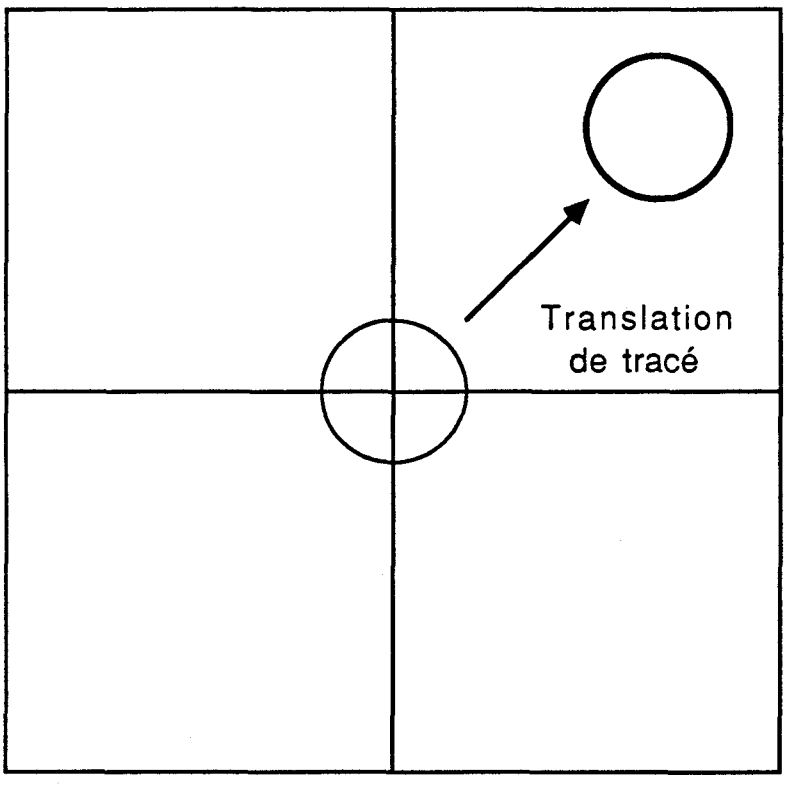


Fig 8

Le tracé d'un cercle de rayon R demanderait un traçage d'environ 4*R*(2 puiss (1/2)) pixels.

Le tracé de cercle du TMS 34010 fait appel à des symétries 1/8, 1/4, 1/2 cadrans pour ce qui est du calcul des valeurs des pixels. Donc çà revient à calculer seulement les valeurs des pixels d'1/8 cadran, et par conséquent en déduire les autres pixels par symétrie.

L'algorithme du tracé de cercle utilise l'instruction cablée "Drav" du TMS 34010: instruction qui permet de tracer un pixel et de passer au suivant en incrémentant les coordonnées du dernier point affiché

(l'instruction "Drav" correspond à deux instructions: (une instruction d'affichage et une addition).

Un cercle de rayon R tracé par un seul processeur demanderait:
(((2puiss(1/2))*(((8 * n) + tc) * (4 * R))) / 8) + ti unités de temps

soit:

(((2 puiss (1/2)) * ((8*n) + tc) * R) / 2) + ti unités de temps

ti: le temps de remplissage et d'initialisation des registres avec les valeurs apropriées (en unités de temps) (environ = 460 unités de temps).

n : la durée d'execution de l'instruction "Drav" (en unités de temps) (n= n aff + n add avec n aff = n add).

Par conséquent, le tracé d'un quart de cercle durerait:

(((2 puiss (1/2)) * ((2 * n) + tc) * R) / 2) + ti' unités de temps

ti': le temps de remplissage et d'initialisation des registres avec les valeurs apropriées (en unités de temps) (environ = 300 unités de temps).

La translation se fait sur un rectangle (pavé) de pixels. Donc pour translater un cercle (où un quart de cercle), on est obligé de trouver le rectangle qui encadrerait le cercle (où le quart de cercle).

C'est l'instruction "Pixblts" du TMS 34010 qui réalise la translation. Le temps d'execution de cette instruction dépend de la taille du pavé de pixels à translater. Il est de:

((4 * m * l) + 5 + ts unités de temps

l : nombre de lignes du pavé de pixels.
m : nombre de pixels par ligne.
t s : temps nécessaire à certaines initialisations (Setup time = 12)

Pour translater un cercle de rayon R, il faudrait translater le carré de côté (2 * R) qui contiendrait le cercle. Donc la translation d'un cercle de rayon R mettrait un temps de:

((4 * (2 * R))) * (2 * R)) + 5 + ts + ti'' unités de temps

soit:

((16 * R)) * R) + 5 + ts + ti''

ti'' : le temps de remplissage et d'initialisation des registres avec les valeurs apropriées (en unités de temps).

Donc le temps de tracé, en parallèle, des quatre quarts de cercle de rayon R par les 4 processeurs et de translation, par le hôte, de ces quatre quarts de cercle est de:

(((2puiss(1/2))*((2*n)+tc)*R)/2)+ti'+5+ts+ti''+((16*R))*R) (1)

Ce qui nous donne:

Ti + ((((2 puiss (1/2))/2)*((2*n)+tc)) + (16*R)) * R) (1)

avec Ti = ti' + 5 + ts + ti''

Alors que le temps du tracé du même cercle (c'est à dire de rayon R), à la position souhaitée, par un seul processeur serait de:

((((2 puiss (1/2))/2)*((8*n)+tc))*R) + ti (2)

On peut négliger ti dans (2) et Ti dans (1). n est de l'ordre de 8. Ce qui nous donne:

(35 + (16 * R)) * R (1)

67 * R (2)

Pour un R < 2, (1) < (2), et pour un R >= 2, (1) >= (2). Donc pour un R >> 2, (1) >> (2).

Ce qui veut dire que, pour un rayon R beaucoup plus grand que 2, le temps du tracé de cercle, par les quatre processeurs, et la translation de ce dernier est beaucoup plus grand que celui du tracé du même cercle par un seul processeur.

A titre d'exemple, le traçage de 10 000 cercles (de centre (290, 210) et de rayon 130) par un seul processeur met un temps d'environ 9 secondes. Alors que le traçage de 10 000 quarts de cercle (au centre de l'écran et de rayon 130) plus leur translation (à leur position effective) met un temps d'environ 42 secondes.

Ceci nous montre, que cette configuration, ne marche pas très bien pour les tracés et les remplissages non centrés au centre de l'écran.

2- DEUSIEME CAS:

Cas où chaque processeur s'occupe de plusieurs pavés de pixels non contigus.

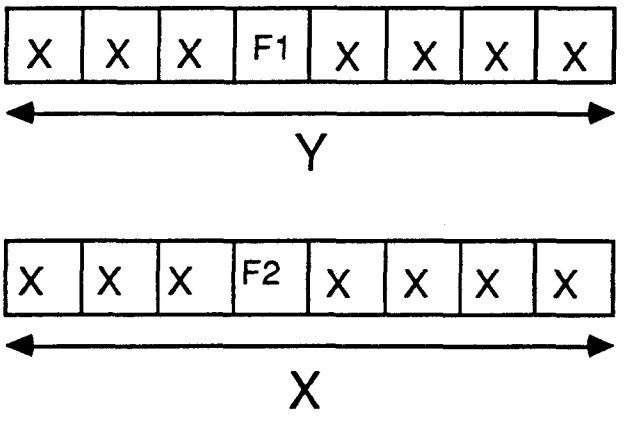


Fig 9

Certains tracés du TMS 34010 font appel à des commandes cablées comme: "Line", "Fill", "Drav",... etc, et à des symétries 1/8, 1/4, 1/2 cadrans pour les calculs des pixels (par exemple: le tracé de cercle).

En adoptant cette configuration, on ne pourrait plus profiter des commandes cablées, qui seraient plus rapides que des commandes logicielles, et des différentes symétries utilisées pour les calculs de pixels par le TMS 34010.

EXEMPLE:

Pour quatre processeurs:

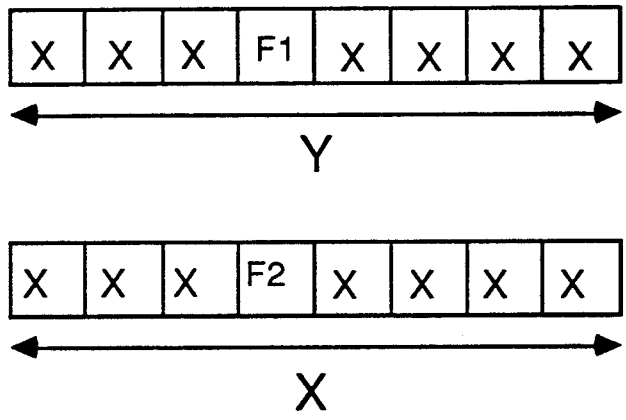


Fig 10

Les fenêtres sont de taille de 256 pixels (16 x 16).
Les pavés sont de taille de 1024 pixels (32 x 32).

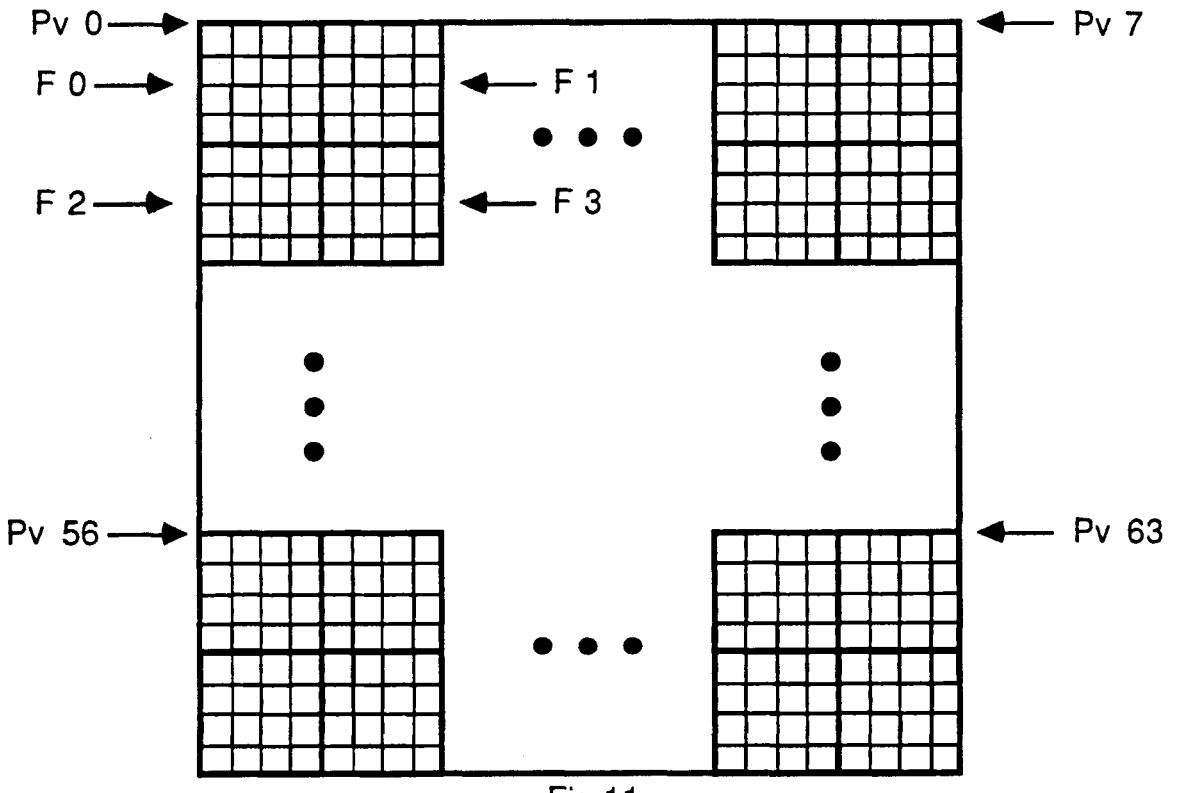


Fig 11

Donnons l'algorithme du tracé de ligne du TMS 34010.

Algorithme:

Lire Xi
Lire Yi
Mettre (Yi, Xi), B2
Lire Xf
Lire Yf
Mettre (Yf, Xf), B10
Mettre ((B10) - (B2)), B10
Si (B10)y = 0 **Alors** aller à **Horiz Fsi**
Si (B10)x = 0 **Alors** aller à **Vert Fsi**
Si (B10)y > 0 **Alors** aller à **bpos Fsi**
Si (B10)x > 0 **Alors** aller à **bneg-apos Fsi**

bneg-aneg: Mettre (-(B10)), B7
Mettre (-1, -1), B11

```
                aller à Comp-b-a
bneg-apos:      Mettre (-(B10)y, (B10)x), B7
                Mettre (-1, 1), B11
                aller à Comp-b-a
bpos:           Si (B10)x > 0 Alors aller à bpos-apos Fsi
bpos-aneg:      Mettre ((B10)y, -(B10)x), B7
                Mettre (1, -1), B11
                aller à Comp-b-a
bpos-apos:      Mettre (B10), B7
                Mettre (1, 1), B11
Comp-b-a:       Mettre (-1, -1), B13
                Mettre (0, (B7)y), B0
                Mettre (0, (B7)x), B10
                Si (B10) >= (B0) Alors aller à a-ge-b Fsi
a-lt-b:         Mettre (B0), B10
                Mettre (0, (B7)x), B0
                Mettre ((B7)x, (B7)y), B7
                Mettre ((B11)y, 0), B12
                Mettre (2 * (B0)), B0
                Mettre ((B0) - (B10)), B0
                Mettre ((B10) + 1), B10
                Si (B11)y < 0 Alors aller à Ligne1 Fsi
Ligne0:         Line 0
                aller à Fin
a-ge-b:         Mettre (0, (B11)x), B12
                Mettre (2 * (B0)), B0
                Mettre ((B0) - (B10)), B0
                Si (B11)y > 0 Alors aller à Ligne0 Fsi
Ligne1:         Line 1
                aller à Fin
Horiz:          Si (B10)x = 0 Alors aller à Pixel Fsi
                Si (B10)x > 0 Alors aller à Do-fill Fsi
                Mettre ((B2) + (B10)), B2
                Mettre (0, -(B10)x), B10
Vert:           Si (B10)y >= 0 Alors aller à Do-fill Fsi
                Mettre ((B10) + (B2)), B2
                Mettre (-(B10)y, 0), B10
Do-fill:        Mettre (B10), B7
                Mettre ((B7)y + 1, (B7)x + 1), B7
                Fill XY
                aller à Fin
```

Pixel: Drav B12, B2
Fin: End

En gros l'algorithme peut se résumer à:

* Saisie des 2 extremités de la ligne à tracer (Yi, Xi) et (Yf, Xf)
* Tester si (Yi, Xi) = (Yf, Xf)
  alors tracer le pixel (Yi, Xi)
  aller à Fin
* Tester si c'est une horizontale
  alors tracer l'horizontale
  aller à Fin
* Tester si c'est une verticale
  alors tracer la verticale
  aller à Fin
* Tester si le tracé se fait dans la direction des y positifs
  alors Line 0
  sinon Line 1
* Fin

On remarque que l'algorithme du tracé de ligne du TMS 34010 utilise des commandes cablées comme: "Line", "Fill", "Drav",... etc. Certaines de ces commandes travaillent sur un ensemble de pixels. Alors que pour tracer une ligne par les quatre processeurs sur cette configuration, on est obligé de connaître les points d'intersection de la ligne avec les différents fenêtres et de travailler au niveau pixel (pour voir dans quelles régions se trouvent les différentes parties de la ligne et par conséquent connaître les processeurs correspondants qui devrait s'en occuper). Et pour connaître les points d'intersection, on est obligé de connaître tous les points de la ligne. Ce qui est contradictoire, puisqu'on ne connaît que les extrémités de la ligne et qu'on est censé tracer cette ligne par les quatre processeurs.

Donc, on sort avec la conclusion que cette configuration ne marche pas pour tout ce qui est tracés de base et remplissage.

3- TROISIEME CAS:

Cas où chaque processeur s'occupe d'un pavé limité à un pixel.

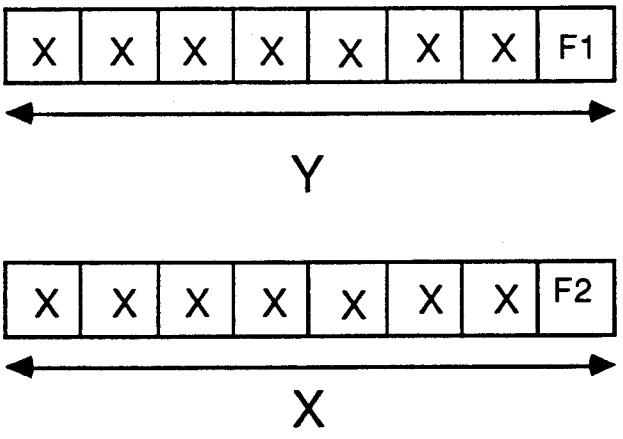


Fig 12

En adoptant cette configuration, on pourrait s'attendre à une augmentation de performances dans tout ce qui n'est pas traité par le TMS 34010 (remplissage , textures, anti-aliassage, ...). En effet, dans cette configuration, on ne se heurte plus aux problèmes de frontières posés par la configuration précédente.

EXEMPLE:

Pour quatre processeurs:

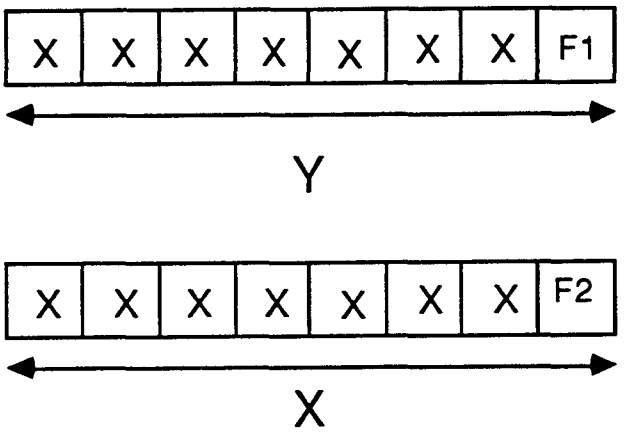


Fig 13

Les fenêtres sont de taille de 1 pixel (1 x 1).
Les pavés sont de taille de 4 pixels (2 x 2).

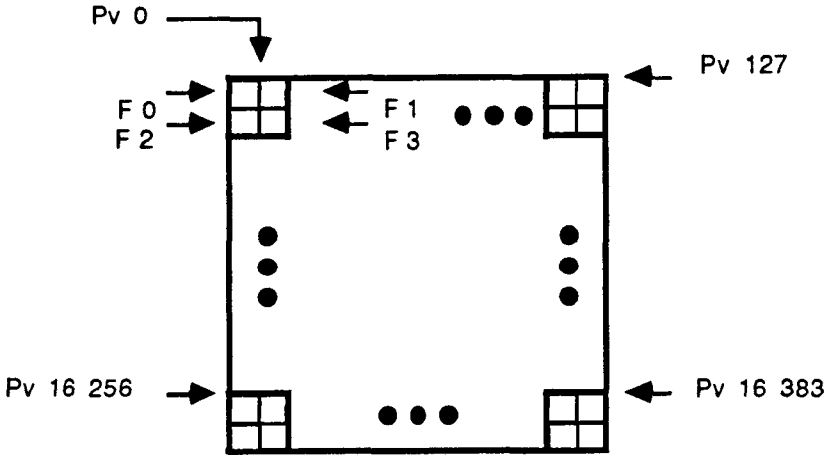


Fig 14

B - LE PARTAGE PAR OBJET:

Le partage par objet (où encore le partage de tâches) consiste à attribuer à chaque processeur un objet. L'ensemble de ces objets, traités en parallèle, forment une partie de l'image à afficher.

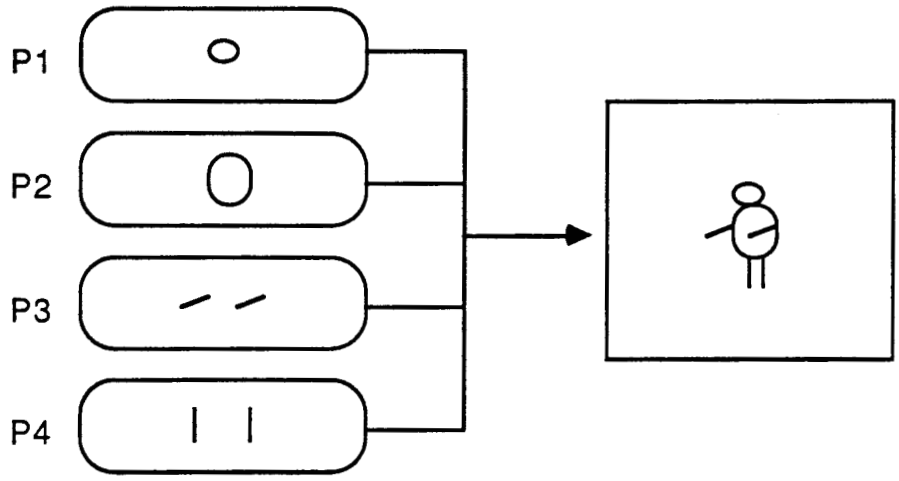


Fig 15

Le problème important concerne les conflits d'accés à la mémoire image.

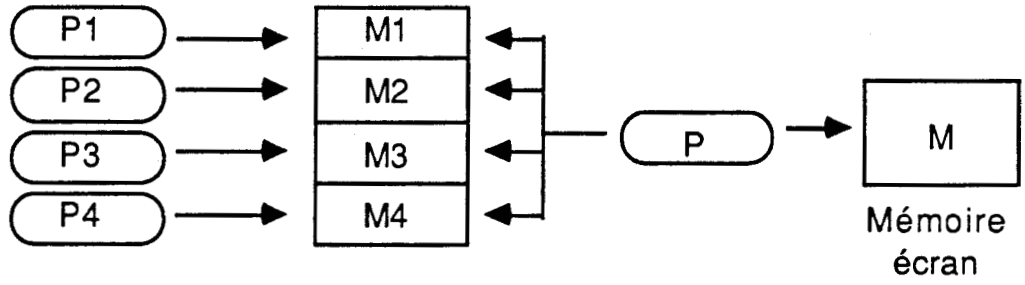


Fig 16

Dans cette configuration, nous utiliserons toutes les performances du TMS 34010 (Transfert de blocs de pixels, Tracé de segment, ...).

IV - CONCLUSION:

En résumé, l'approche pipeline est à rejeter puisque cette approche permet le regroupement de processeurs, chacun executant une tâche élémentaire, alors que le processeur spécialisé offre déjà toutes ces opérations. On retiendra du partage géométrique, le cas où le pavé est limité à un pixel qui pourrait donner de bons résultats pour toutes les opérations non traitées par le processeur spécialisé. On essayera d'approfondir l'approche du partage par objet qui utiliserait toute la puissance des processeurs spécialisés.

CHAPITRE IV

LE PARTAGE GEOMETRIQUE

SOMMAIRE

I - INTRODUCTION:

Dans ce chapitre, on va étudier, pour cette configuration, le tracé de segment et le remplissage. Pour le tracé de segment, il faudrait réécrire l'algorithme puisque celui du TMS 34010 utilise la commande cablée "Line", et par conséquent il est inutilisable en parallèle. Quand au remplissage, vu que le TMS 34010 ne traite que le cas particulier du remplissage d'un pavé rectangulaire de pixels, il nous faudrait trouver l'algorithme approprié pour le remplissage dans le cas général c'est à dire le cas d'une tâche polygonale quelconque.

II - LE TRACE DE SEGMENT:

Dans ce paragraphe, on va aborder le tracé de segments qui est considéré comme l'outil le plus important de la réalisation d'un logiciel graphique. Le problème du tracé de lignes revient à trouver une approximation telle que l'oeil puisse reconstruire le segment de droite représenté. Plusieurs algorithmes ont été proposés ([Bre 65], [Ear 77], [Luc 74]) dont la caractéristique principale est d'éviter les opérations de multiplication et de division et de travailler en entiers. Deux algorithmes sont particulièrement intéréssants, celui de Lucas et celui de Bresenham.

Algorithme de Lucas:

Dans celui de Lucas, pour engendrer un segment, nous nous ramenons au premier octant. Et comme nous nous déplaçons, systématiquement à chaque pas, suivant l'axe des x, nous évaluons l'erreur commise sur les ordonnées si nous ne modifions pas y. Afin de la minimiser, il suffit de modifier y (en ajoutant 1) lorsque cette erreur atteindra ou dépassera l'unité.

Algorithme de Bresenham:

Dans celui de Bresenham, pour générer un segment, nous déterminons, à chaque pas, le point le plus proche du segment idéal.

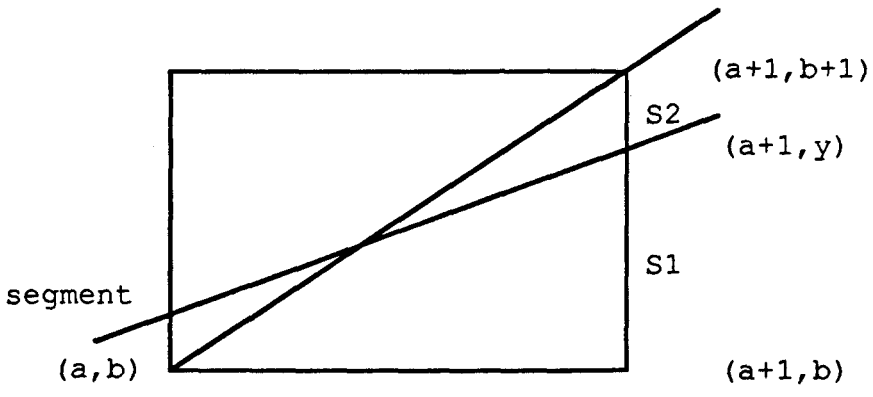


Fig 1

C'est pourquoi, nous évaluons de façon récurrente la différence (S1-S2) (voir Fig 1) en nous situant dans le premier octant:
Si (S1-S2)<0 nous effectuons un mouvement axial, sinon nous effectuons un mouvement diagonal.

On définit un segment de droite par la donnée des coordonnées de ses extrémités:

Fig 2

1 - L'algorithme du TMS 34010:

L'algorithme utilisé dans le TMS 34010 est l'algorithme de Bresenham:

§ saisir première extremité (xi, yi)
§ saisir deusième extremité (xf, yf)
§ a = xf - xi
§ b = yf - yi
§ si (a = 0) et (b = 0) alors afficher le pixel (xi, yi)

```
                    sinon si (a < 0) alors   a = - a
                                             Dx1 = 1
                                     sinon
                                     finsi
                         si (b < 0)  alors   b = - b
                                             Dy1 = 1
                                     sinon
                                     finsi
                         si (b <= a) alors   Dx2 = Dx1
                                     sinon   c = a
                                             b = c
                                             Dy2 = Dy1
                                     finsi
                    finsi
§ Er = (2 * b) - a
§ Ed = (2 * b) - (2 * a)
§ Ehv = 2 * b
§ c = a + 1
§ tester:
  si c'est une horizontale
  alors tracer l'horizontale (fill xy)
  sinon tester:     si c'est une verticale
                    alors tracer la verticale (fill xy)
                    sinon tester:   si  le tracé se fait
                                        dans le sens des
                                        y positifs
                                    alors tracer la ligne
                                          (line 0)
                                    sinon tracer la ligne
                                          (line 1)
                                    finsi
                    finsi
  finsi
§ fin
```

L'algorithme du tracé de lignes du TMS 34010 est un algorithme qui utilise la commande cablée "Line". Donc pour pouvoir paralléliser

l'algorithme du tracé de segments, il faut d'abord réécrire l'algorithme du TMS 34010 sans utiliser la commande cablée "Line" et puis seulement après s'intérésser à la parallélisation.

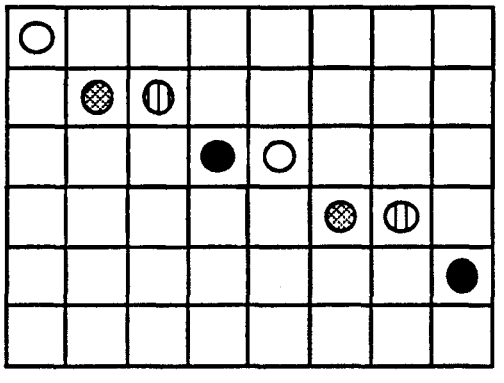
L'algorithme réécrit est le suivant:

```
§ saisir première extremité (xi, yi)
§ saisir deusième extremité (xf, yf)
§ a = xf - xi
§ b = yf - yi
§ si (a = 0) et (b = 0) alors afficher le pixel (xi, yi)
                        sinon si (a < 0)  alors   a = - a
                                                  Dx1 = - 1
                                          sinon   Dx1 = 1
                                          finsi
                              si (b < 0)  alors   b = - b
                                                  Dy1 = - 1
                                          sinon   Dy1 = 1
                                          finsi
                              si (b <= a) alors   Dx2 = Dx1
                                                  Dy2 = 0
                                          sinon   c = a
                                                  a = b
                                                  b = c
                                                  Dx2 = 0
                                                  Dy2 = Dy1
                                          finsi
  finsi
§ Er = (2 * b) - a
§ Ed = (2 * b) - (2 * a)
§ Ehv = 2 * b
§ c = a + 1
§ tant que (c > 0) faire afficher le pixel (xi, yi)
                        teste si (Er>=0)  alors   Er = Er + Ed
                                                  xi = xi + Dx1
                                                  yi = yi + Dy1
                                          sinon   Er = Er + Ehv
                                                  xi = xi + Dx2
                                                  yi = yi + Dy2
                                          finsi
                      c = c - 1
```

§ finto
§ fin

2 - Application à la configuration (2x2):

Dans une configuration à quatre processeurs (2x2), la parallélisation revient à faire tracer les pixels du segment par les processeurs qui s'en occupent. Essayons de prendre un exemple:



les pixels ○ : correspondent aux pixels du processeur P1
les pixels ● : correspondent aux pixels du processeur P2
les pixels ◍ : correspondent aux pixels du processeur P3
les pixels ⊗ : correspondent aux pixels du processeur P4

Fig 3

la parallélisation du tracé revient à essayer de faire tracer (en parallèle):

les pixels 1 et 5 du segment par le processeur 1
les pixels 2 et 6 du segment par le processeur 4
les pixels 3 et 7 du segment par le processeur 3
les pixels 4 et 8 du segment par le processeur 2

On remarque qu'en utilisant un Bresenham à pas de 2 et une condition sur l'erreur calculée, afin de décider de l'affichage du pixel ou non, on devrait arriver à cette parallélisation.

Intéréssons nous maintenant au calcul d'erreur.

L'erreur commise lors d'un tracé de segment de droite dans une configuration à 1 seul processeur (1x1) est:

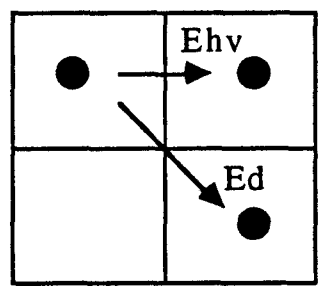


Fig 4

avec b = Min (| xf - xi |, | yf - yi |)
a = Max (| xf - xi |, | yf - yi |)

Ehv = 2 * b
Ed = (2 * b) - (2 * a)

et l'erreur initiale Ei = (Ehv + Ed) / 2 = (2 * b) - a

En transposant ceci sur une configuration parallèle à quatre processeurs (2x2), on trouve:

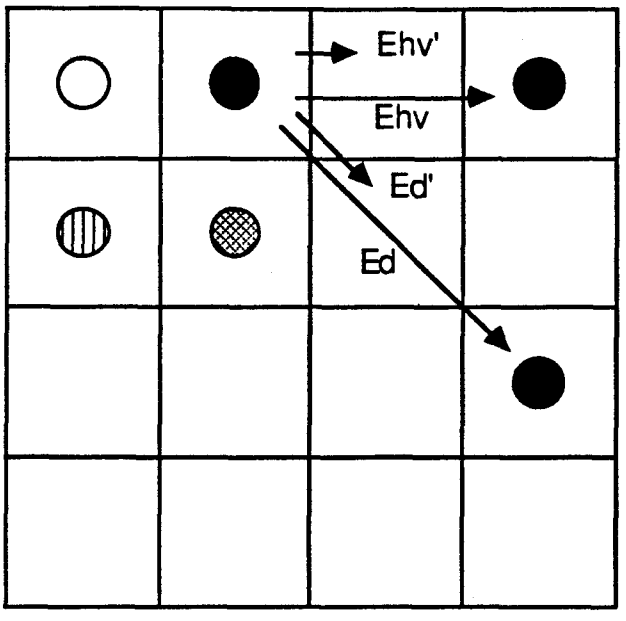


Fig 5

avec $Ei' = (Ehv' + Ed') / 2$

Reste à exprimer Ehv' et Ed' en fonction de Ehv et Ed, qui représentent respectivement l'erreur horizontale-verticale et l'erreur diagonale dans une configuration à un seul processeur (1x1).

Du fait de l'utilisation d'un Bresenham à pas de 2, j'ai pris pour:

$$Ehv' = Ehv / 2 = b - a$$
$$Ed' = Ed / 2 = b$$
$$Ei' = Ei / 2 = ((2 * b) - a) / 2$$

La condition qui nous permet d'afficher où non le pixel calculé est:

$$(Ed / 2) =< Erreur < (Ehv / 2)$$

Quand cette condition est vérifiée on affiche le pixel sinon on n'affiche pas et on passe au calcul du pixel suivant.

Essayons de nous intérésser à l'erreur initiale:

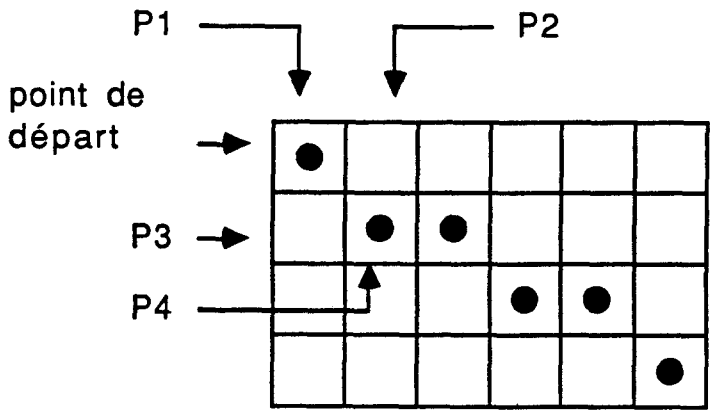


Fig 6

Le processeur par lequel passe le point de départ du segment est numéroté 1. Celui qui est à droite (où à gauche) de ce point est numéroté 2, et celui qui est juste en dessous (où au dessus) est numéroté 3.

CAS 1:

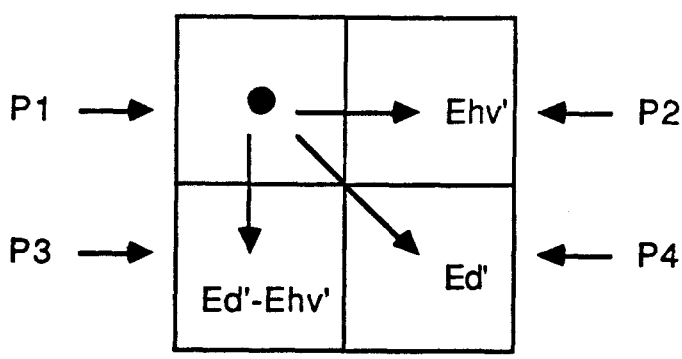


Fig 7

$$|xf - xi| >= |yf - yi|$$

Dans ce cas l'erreur initiale du processeur:

P1 est Ei1 = (Ehv' + Ed') / 2
P2 est Ei2 = Ei1 + Ehv'
P3 est Ei3 = Ei1 + Ed' - Ehv'
P4 est Ei4 = Ei1 + Ed'

D'où Ei1 = (b + b - a) / 2 = ((2 * b) - a) / 2
Ei2 = (((2 * b) - a) / 2) + b = ((4 * b) - a) / 2
Ei3 = (((2 * b) - a) / 2) + b - a - b = ((2 * b) - (3 * a)) / 2
Ei4 = (((2 * b) - a) / 2) + b - a = ((4 * b) - (3 * a)) / 2

CAS 2:

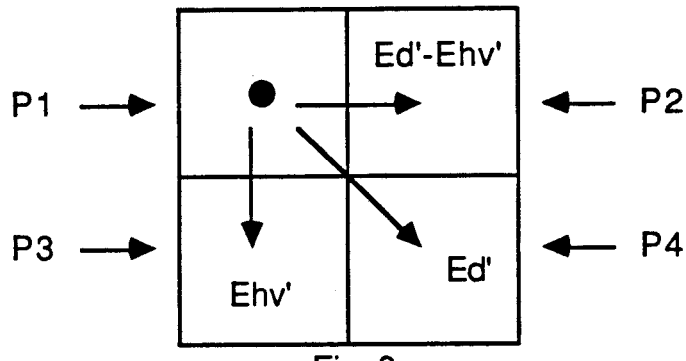


Fig 8

$|xf - xi| < |yf - yi|$

Dans ce cas l'erreur initiale du processeur:

P1 est Ei1 = (Ehv' + Ed') / 2
P2 est Ei2 = Ei1 + Ed' - Ehv'
P3 est Ei3 = Ei1 + Ehv'
P4 est Ei4 = Ei1 + Ed'

D'où Ei1 = (b + b - a) / 2 = ((2 * b) - a) / 2
Ei2 = (((2 * b) - a) / 2) + b - a - b = ((2 * b) - (3 * a)) / 2

Ei3 = (((2 * b) - a) / 2) + b = ((4 * b) - a) / 2
Ei4 = (((2 * b) - a) / 2) + b - a = ((4 * b) - (3 * a)) / 2

Comme on utilise un Bresenham à pas de 2, donc, pour un processeur, on calcule au maximum la moitié des points du segment. C'est pourquoi le compteur est initialisé à la moitié de la valeur du compteur utilisé dans le tracé de segment, dans le cas d' une configuration à 1 seul processeur (1x1).

On a pu remarquer que dans certains cas, on calcule et on affiche des pixels qui sont en dehors du tracé de segment. Ceci s'explique par le fait que dans certains cas: un des 4 processeurs ne devrait calculer qu'un nombre de pixels inférieur à celui initialisé (c'est à dire la valeur du compteur), ce qui fait qu'on calcule des points en plus et qui sont en dehors du segment. Et si le calcul d'erreur correspondant vérifie la condition d'affichage, on affichera le point.

Exemple de cas:

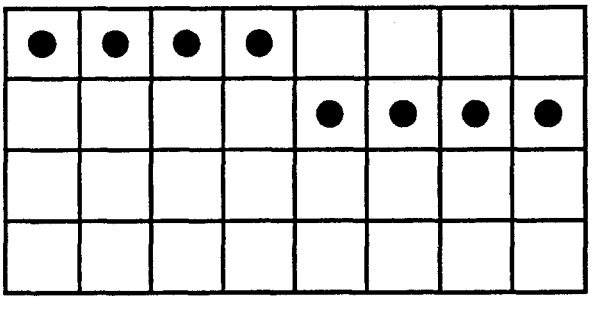


Fig 9

Pour remédier à ce problème, on introduit une condition dans l'algorithme. Cette condition est le test du point calculé avec le point de l'extremité du segment, et dés que la condition n'est plus vérifiée, on arrête le calcul.

L'algorithme parallélisé du tracé de segment sur une configuration à 4 processeurs (2x2) est le suivant:

```
§ saisir première extremité (xi, yi)
§ saisir deusième extremité (xf, yf)
§ saisir le n° du processeur (e)
§ a = xf - xi
§ b = yf - yi
§ si (a = 0) et (b = 0) alors afficher le pixel (xi, yi)
                        sinon si (a < 0)  alors   a = - a
                                                  Dx1 = - 2
                                          sinon   Dx1 = 2
                                          finsi
                              si (b < 0)  alors   b = - b
                                                  Dy1 = - 2
                                          sinon   Dy1 = 2
                                          finsi
                              si (b <= a) alors   Dx2 = Dx1
                                                  Dy2 = 0
                                          sinon   c = a
                                                  a = b
                                                  b = c
                                                  Dx2 = 0
                                                  Dy2 = Dy1
                                          finsi
  finsi
§ Er = ((2 * b) - a) / 2
§ Ed = (2 * b) - (2 * a)
§ Ehv = 2 * b
§ c = (a / 2) + 1
§ Cd = b - a
§ Chv = b
§ tester:
  si          (Er est impair) et (Er < 0)
  alors       Er = Er - 1
  finsi
§ cas e dans:
  2: tester:  si      on est dans le cas  I xf - xi I < I yf - yi I
              alors   Er = Er - Chv + Cd
              sinon   Er = Er + Chv
              finsi
  3: tester:  si      on est dans le cas  I xf - xi I < I yf - yi I
              alors   Er = Er + Chv
              sinon   Er = Er - Chv + Cd
```

```
                finsi
   4:  Er = Er + Cd
   fincas
§  tant que (c > 0) et ((xi,yi) se trouve sur le segment) faire
       tester   si      (Cd <= Er) et (Er < Chv)
                alors   afficher le pixel (xi, yi)
                finsi
       tester   si      (Er >= 0)
                alors   Er = Er + Ed
                        xi = xi + Dx1
                        yi = yi + Dy1
                sinon   Er = Er + Ehv
                        xi = xi + Dx2
                        yi = yi + Dy2
                finsi
       c = c - 1
§  fintq
§  fin
```

3 - Application à la configuration (4x4):

Le passage d'une configuration parallèle, à 4 processeurs (2x2), à une configuration parallèle, à 16 processeurs (4x4), se fait sans problème. Dans cette configuration, au lieu d'utiliser un Bresenham à pas de 2, on utilise un Bresenham à pas de 4. La seule différence entre les configurations est la suivante:

L'erreur commise lors d'un tracé de segment de droite dans une configuration à 16 processeurs (4x4) est:

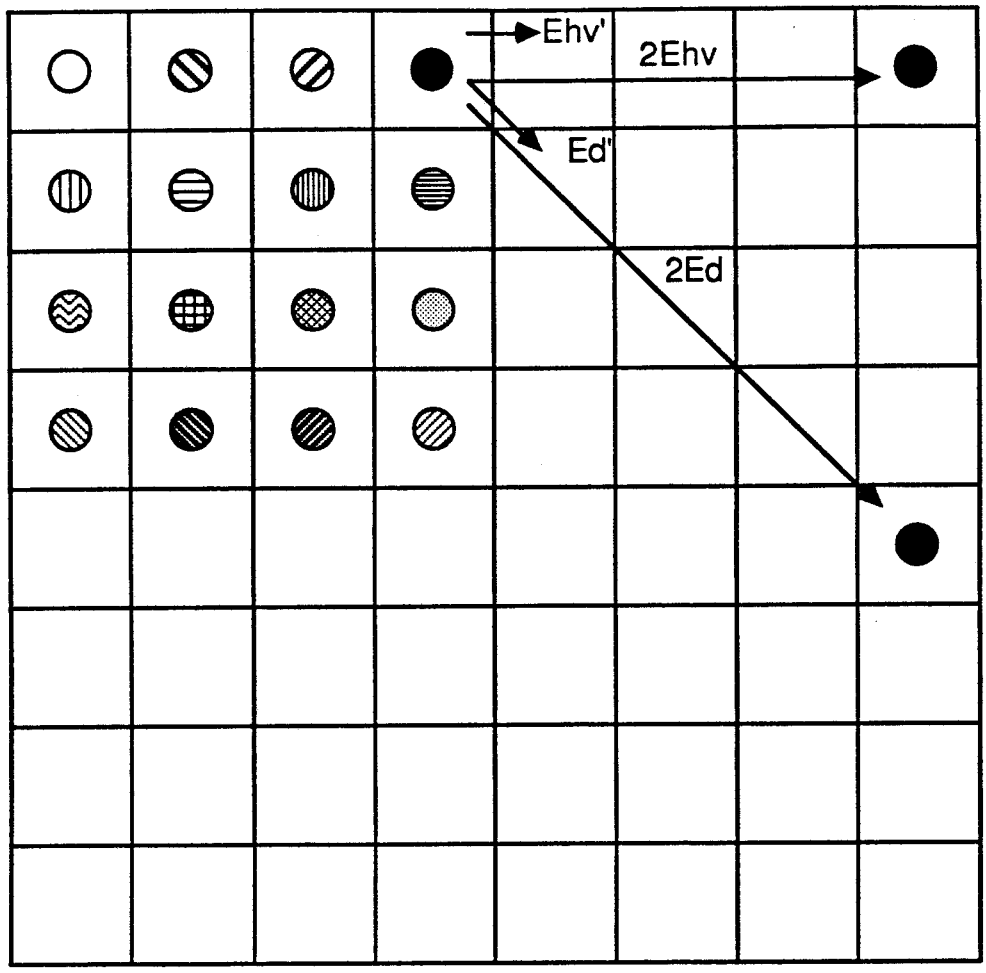


Fig 10

avec

Ehv' = (2 * Ehv) / 4 = Ehv / 2 = b
Ed' = (2 * Ed) / 4 = Ed / 2 = b - a
Ei' = (Ehv' + Ed') / 2 = (Ehv + Ed) / 4 = Ei / 2

d'où

Ei' = ((2 * b) - a) / 2

Sinon pour le reste, il se déduit sans aucun problème de la configuration à 4 processeurs (2x2):

La condition d'affichage est la même c'est à dire:

(Ed / 2) =< Erreur < (Ehv / 2)

Le calcul d'erreur initiale se calcule de la même façon que dans le cas d'une configuration à 4 processeurs (2x2).

Le compteur du nombre de points à calculer est initialisé à la moitié de celui utilisé dans l'algorithme de la configuration à 4 processeurs (2x2) (puisqu'on utilise un Bresenham à pas de 4).

Là aussi, on utilise le test du point calculé avec le point de l'extremité du segment à tracer.

L'algorithme parallélisé du tracé de segments sur une configuration à 16 processeurs (4x4) est le suivant:

§ saisir première extremité (xi, yi)
§ saisir deusième extremité (xf, yf)
§ saisir le n° du processeur (e)
§ a = xf - xi
§ b = yf - yi
§ si (a = 0) et (b = 0) alors afficher le pixel (xi, yi)

```
                    sinon si (a < 0)  alors   a = - a
                                              Dx1 = - 4
                                      sinon   Dx1 = 4
                                      finsi
                          si (b < 0)  alors   b = - b
                                              Dy1 = - 4
                                      sinon   Dy1 = 4
                                      finsi
                          si (b <= a) alors   Dx2 = Dx1
                                              Dy2 = 0
                                      sinon   c = a
                                              a = b
                                              b = c
                                              Dx2 = 0
```

```
                                                    Dy2 = Dy1
                                                finsi
    finsi
§   Er = ((2 * b) - a) / 2
§   Ed = (2 * b) - (2 * a)
§   Ehv = 2 * b
§   c = (a / 4) + 1
§   Cd = b - a
§   Chv = b
§   tester:
    si          (Er est impair)  et  (Er < 0)
    alors       Er = Er - 1
    finsi
§   cas e dans:
    2:  tester:  si      on est dans le cas  I xf - xi I < I yf - yi I
                 alors         Er = Er - Chv + Cd
                 sinon         Er = Er + Chv
                 finsi
    3:  tester:  si      on est dans le cas  I xf - xi I < I yf - yi I
                 alors  Er = Er + (2 * Cd) - (2 * Chv)
                 sinon  Er = Er + (2 * Chv)
                 finsi
    4:  tester:  si      on est dans le cas  I xf - xi I < I yf - yi I
                 alors  Er = Er + (3 * Cd) - (3 * Chv)
                 sinon  Er = Er + (3 * Chv)
                 finsi
    5:  tester:  si      on est dans le cas  I xf - xi I < I yf - yi I
                 alors  Er = Er + Chv
                 sinon  Er = Er - Chv + Cd
                 finsi
    6:  Er = Er + Cd
    7:  tester:  si      on est dans le cas  I xf - xi I < I yf - yi I
                 alors  Er = Er + (2 * Cd) - Chv
                 sinon  Er = Er + Chv + Cd
                 finsi
    8:  tester:  si      on est dans le cas  I xf - xi I < I yf - yi I
                 alors  Er = Er + (3 * Cd) - (2 * Chv)
                 sinon  Er = Er + (2 * Chv) + Cd
                 finsi
    9:  tester:  si      on est dans le cas  I xf - xi I < I yf - yi I
                 alors  Er = Er + (2 * Chv)
```

```
                    sinon  Er = Er - (2 * Chv) + (2 * Cd)
                    finsi
   10:  tester:     si     on est dans le cas  I xf - xi I < I yf - yi I
                    alors  Er = Er + Chv + Cd
                    sinon  Er = Er - Chv + (2 * Cd)
                    finsi
   11:  Er = Er + (2 * Cd)
   12:  tester:     si     on est dans le cas  I xf - xi I < I yf - yi I
                    alors  Er = Er + (3 * Cd) - Chv
                    sinon  Er = Er + Chv + (2 * Cd)
                    finsi
   13:  tester:     si     on est dans le cas  I xf - xi I < I yf - yi I
                    alors  Er = Er + (3 * Chv)
                    sinon  Er = Er - (3 * Chv) + (3 * Cd)
                    finsi
   14:  tester:     si     on est dans le cas  I xf - xi I < I yf - yi I
                    alors  Er = Er + (2 * Chv) + Cd
                    sinon  Er = Er - (2 * Chv) + (3 * Cd)
                    finsi
   15:  tester:     si     on est dans le cas  I xf - xi I < I yf - yi I
                    alors  Er = Er + Chv + (2 * Cd)
                    sinon  Er = Er - Chv + (3 * Cd)
                    finsi
   16:  Er = Er + (3 * Cd)
   fincas
§  tant que (c > 0) et ((xi,yi) se trouve sur le segment) faire
   tester:          si     (Cd <= Er) et (Er < Chv)
                    alors  afficher le pixel (xi, yi)
                    finsi
   tester:          si     (Er >= 0)
                    alors  Er = Er + Ed
                           xi = xi + Dx1
                           yi = yi + Dy1
                    sinon  Er = Er + Ehv
                           xi = xi + Dx2
                           yi = yi + Dy2
                    finsi
   c = c - 1
§  fintq
§  fin
```

4 - Performances:

Le traçage de dix mille lignes (ligne allant du point (10, 300) au point (450, 15)) par l'algorithme non parallélisé et réécrit pour la configuration à 1 seul processeur (1x1) (n'utilisant pas la commande cablée "line" du TMS 34010) met un temps d'environ 12 secondes.

Le même traçage mais, cette fois çi, effectué par l'algorithme parallélisé correspondant à la configuration à 4 processeurs (2x2) met un temps d'environ 12 secondes. Ce qui voudrait dire qu'on ne gagne pas du tout en temps. Ceçi s'explique par le fait qu'on a introduit plusieurs tests (nécessaires) dans notre algorithme parallélisé, comme: le test de l'erreur pour permettre l'affichage ou non du pixel, le test pour l'initialisation de l'erreur, et le test pour voir si le pixel calculé se trouve toujours sur le segment.

Alors que le même traçage, réalisé par l'algorithme parallélisé correspondant à la configuration à 16 processeurs (4x4) met un temps d'environ 8 secondes. Ce qui voudrait dire qu'avec cet algorithme, on gagne un tiers du temps d'execution, et par conséquent, les tests, introduits dans l'algorithme, commençent à peser moins lourd.

III - LE REMPLISSAGE:

A - INTRODUCTION:

Dans le remplissage de tâches, on distingue trois types de traitements:

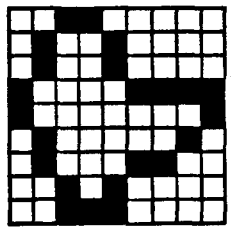
- Coloriage
- Remplissage proprement dit
- hachurage

1 - Coloriage:

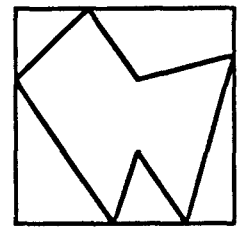
Dessiner point à point le contour d'une tâche puis peindre d'une couleur en utilisant un point intérieur: germe ("seed").

2 - Remplissage:

Remplir une tâche dont le contour est défini soit dans l'espace utilisateur soit préinscrit point par point dans une mémoire d'image.



Contour défini point par point dans la mémoire image



Contour polygonal défini dans l'espace utilisateur

Fig 11

L'algorithme de remplissage se décompose en deux phases:

- Préparation du contour: constitution d'une liste d'arêtes lorsqu'on raisonne dans l'espace utilisateur, où inscription et codage du contour dans une mémoire d'image (représentant l'écran).

- Remplissage de la tâche avec la couleur désirée.

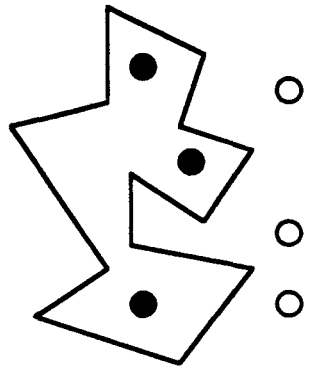
3 - Hachurage:

Utilisation de certaines techniques de remplissage ligne par ligne et en particulier les algorithmes de "suivi de contour".

Nous ne nous intérésserons ici qu'à la partie remplissage.

B - REMPLISSAGE DE TACHES:

Pour executer l'algorithme, il faudrait que l'on sache si on est à l'intérieur où à l'extérieur de la zône à remplir: d'où l'utilisation de l'algorithme du "contrôle de parité".



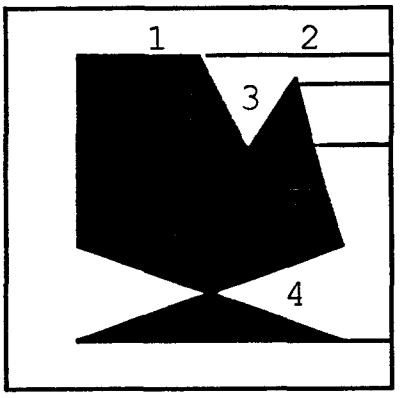
● : Point intérieur (nombre impair d'intersections)

○ : Point extérieur (nombre pair d'intersections)

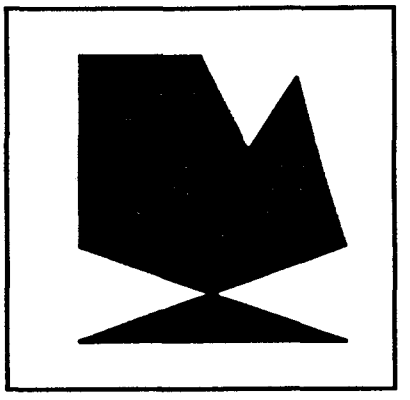
Fig 12

Mais l'application de cet algorithme, donne des résultats faux dans certains cas particuliers:

Exemple:



Erreurs de remplissage dues au "Contrôle de parité"



Remplissage corresct

1- arêtes horizontales
2- sommets
3- recouvrements d'arêtes
4- points doubles

Fig 13

De ce fait, d'autres algorithmes ont été proposés pour remédier aux insuffisances de cet algorithme, et qu'on peut regrouper dans deux classes :

- Les techniques de balayage ligne par ligne, utilisées surtout dans le cas du codage de contour dans la mémoire image.

- Les techniques de suivi de contour, utilisées surtout dans le cas d'une description structurée de contour.

Le TMS 34010 ne propose pas d'algorithmes de remplissage où du moins de remplissage de tâches quelconques. Il présente une instruction cablée: "Fill" qui permet le remplissage d'une zône

rectangulaire. Nous ici, on propose un algorithme plus général, puisqu'il traite à priori toutes les tâches polygonales.

C - REMPLISSAGE DANS L'ESPACE UTILISATEUR:

L'algorithme utilisé est l'algorithme de suivi de contour par complémentation.

Tout d'abord, on essaye de trouver le point le plus à droite de la tâche, qui nous servirait comme limite au moment du remplissage au lieu de remplir jusqu'au bord droit de l'écran.

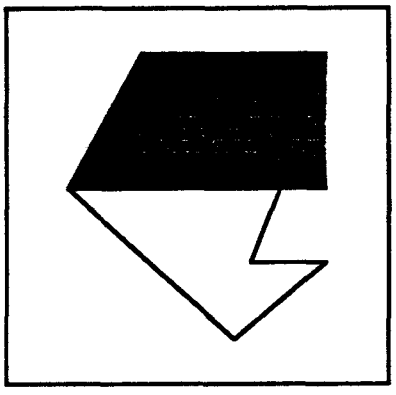
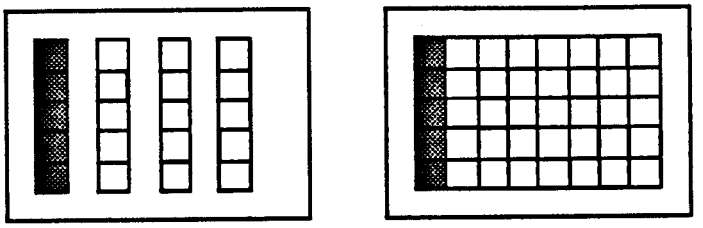


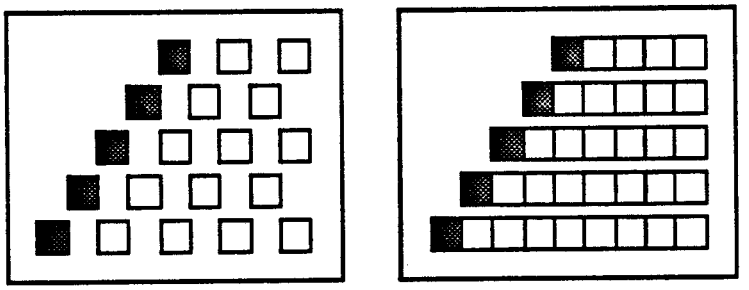
Fig 14

Après, on essaye de trouver le point le plus haut le plus à gauche du contour, qui va être notre point de départ. On prend comme sens de parcours du contour le sens contraire à celui des aiguilles d'une montre. Ensuite, on essaye de tracer le contour arête par arête. Pour le tracé de l'arête, on utilise un Bresenham de pas n pour une configuration nxn processeurs. Tout en traçant un point d'une arête, on trace, par complémentation, la ligne correspondante et ce jusqu'au bord droit.

Suite à la configuration, il s'avère qu'on a des trous dans certains cas:



Arête verticale



Arête diagonale

Fig 15

Pour pallier çà, on essaye de dédoubler l'arête:

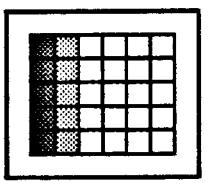


Fig 16

tout en interdisant la situation suivante:

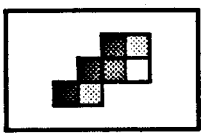


Fig 17

c'est à dire, là ou il n'y a pas de trous on ne dédouble pas (on n'affiche pas le pixel correspondant) puisqu'on utilise un algorithme de complémentation.

Ensuite, il fallait traiter le cas des horizontales:

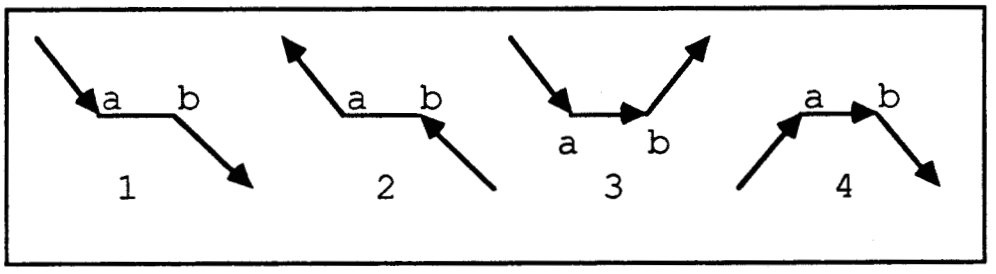


Fig 18

Dans les cas (1) et (2), le traitement de l'horizontale ab se résume au traitement du point b seulement. Tandis que dans les cas (3) et (4), il n'y a pas de traitement à faire, autrement dit, on ignore l'horizontale ab.

Après, il faut traiter le cas des points singuliers:

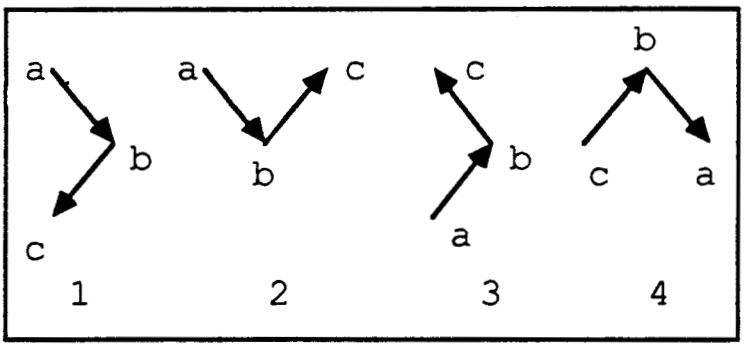


Fig 19

Dans les cas (2) et (4), il n'y a rien à faire: on traite normalement le segment ab puis seulement après le segment bc. A l'inverse, dans

les cas (1) et (3), on traite d'abord le segment ab, après le point b et seulement après, le segment bc.

Après avoir résolu les problèmes de trous, horizontales, points singuliers, on obtient un remplissage correct de la tâche, et ceci dans des temps trés performants. En effet, les résultats de la simulation, nous ont montré que le remplissage, en parallèle, de 100 fois, d'une tâche polygonale d'environ 100 000 pixels, se fait en moins de 20 secondes. Alors que le remplissage, par un seul processeur, de 100 fois, de la même tâche, met un temps d'environ 62 secondes.

IV - CONCLUSION:

Le partage géométrique (cas du pavé limité à un pixel) nous permet d'obtenir de bonnes performances pour tous les tracés non traités par le TMS 34010 (remplissage, anti-aliassage, ...). L'inconvénient c'est que dans cette configuration, on n'utilise pas au maximum les performances des processeurs graphiques spéçialisés (TMS 34010), comme le tracé de ligne cablé, le transfer de blocs de pixels, ...

CHAPITRE V

LE PARTAGE OBJET

SOMMAIRE

I - INTRODUCTION:

Le partage objet consiste en une décomposition d'une scène en plusieurs objets élémentaires (segment, cercle, rectangle, triangle), ou éventuellement en groupements de ces objets élémentaires, chacun étant traité par un processeur. Donc, à un instant donné, plusieurs objets élémentaires seront traités en parallèle.

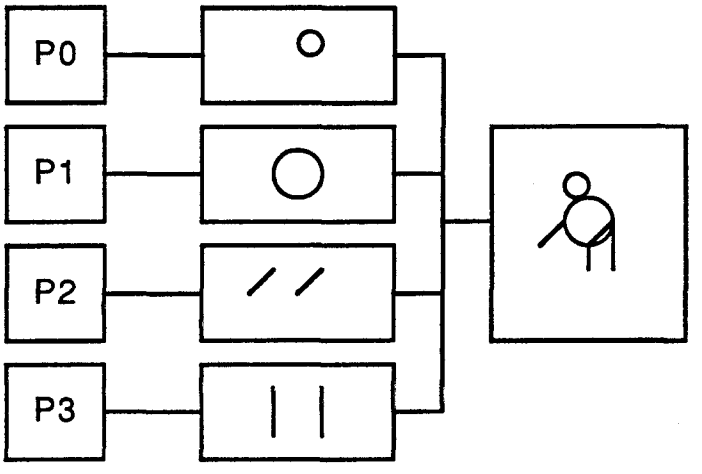


Fig 1

Parmi les caractéristiques fondamentales de cette solution on cite la facilité des traitements appliqués à l'objet. En effet, les traitements appliqués à l'objet tels que translation, rotation, remplissage, ..., sont plus faciles en utilisant les éléments simples résultant qu'en utilisant l'objet lui-même:

* Un objet bien choisi c'est à dire simple, se caractérise par une description simple dans l'espace de construction.

Exemple:

Un segment S se caractérise par la donnée des coordonnées de ces deux extrémités (2 triplets): (x1, y1, z1), (x2, y2, z2).

La manipulation de cet objet repose donc sur des calculs appliqués aux paramètres de la description.

Exemple:

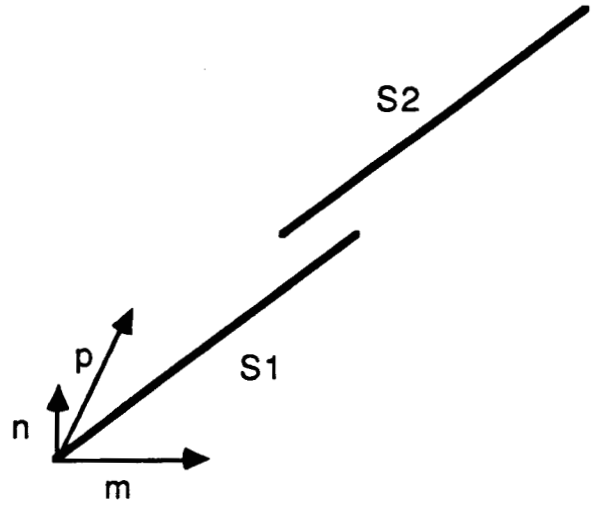


Fig 2

m : translation suivant l'axe x
n : translation suivant l'axe y
p : translation suivant l'axe z
S1 : segment à translater
S2 : segment translaté

Translation du segment:

x1 -> x1+m x2 -> x2+m
y1 -> y1+n y2 -> y2+n
z1 -> z1+p z2 -> z2+p

Donc la translation est réalisée en effectuant les six additions. Cette opération peut donc être confiée à un seul processeur pour tous les objets.

* La projection de l'objet dans l'espace d'affichage (pixels) est beaucoup plus lourde en calcul (Bresenham sur segment s par exemple). C'est donc elle qu'il faut paralléliser.

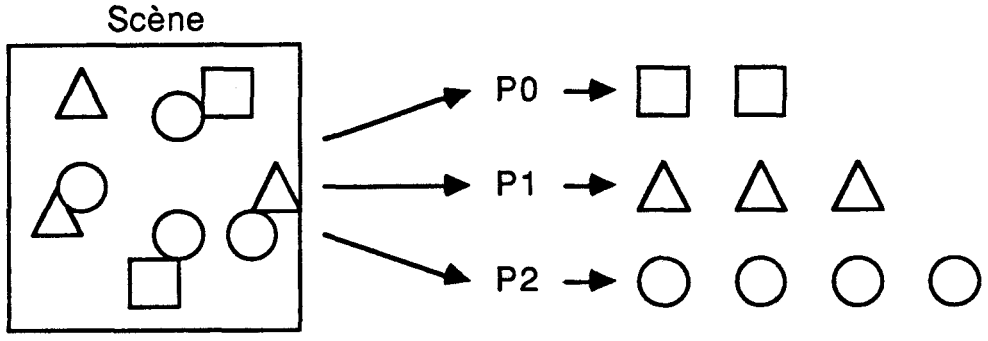
II - ETUDE DU PARTAGE OBJET:

Pour la réalisation du partage objet, on distingue trois possibilités:

* partage suivant le type d'objets (processeurs typés).
* partage suivant les zônes d'objets.
* partage suivant l'arborescence d'objets.

1 - PARTAGE SUIVANT LE TYPE D'OBJETS:

Dans cette configuration, les processeurs objets sont typés, c'est à dire que chaque processeur objet traite un ou plusieurs types d'objets bien détérminés (exemple: les triangles). Tous les objets appartenant au même type seront traités par le même processeur objet.



Partage suivant le type d'objets

Fig 3

Parmi les problèmes qui apparaissent en étudiant cette configuration, on trouve:

a - Problème d'équilibrage de charges:

Du fait que chaque processeur objet s'occupe d'un type d'objet, le taux d'utilisation d'un processeur objet d'un type donné dépend directement de la scène. Si, par exemple, on se trouve en présence d'une scène composée seulement d'un type d'objet donné (triangle par exemple), la configuration parallèle se comportera comme une configuration séquentielle constituée d'un seul processeur actif. Dans cette configuration, dans la majorité des cas, il y aura certains processeurs objets qui vont fonctionner plus que d'autres. Donc, en adoptant cette configuration, il en résultera un problème d'équilibrage de charges.

b - Vitesse:

Du fait du non équilibrage de charges entre les processeurs objets, le temps d'execution d'une scène donnée dépend du processeur objet qui a le plus d'objets élémentaires à traiter. Et comme, dans la plus part des cas, certains processeurs objets travaillent plus que d'autres, cette configuration n'est pas la plus performante au niveau temps d'éxecution. Prenant l'exemple du cas particulier où la scène est composée d'un seul type d'objet, le temps d'execution de cette scène, dans ce cas là, est celui d'une configuration séquentielle composée d'un seul processeur (puisque, dans ce cas là, il n'y a qu'un seul processeur objet qui va travailler). Par contre, la spécialisation d'un processeur permet d'espérer qu'il se montrera plus rapide dans l'execution de sa tâche.

c - Impossibilité de traiter un groupement d'objets élémentaires:

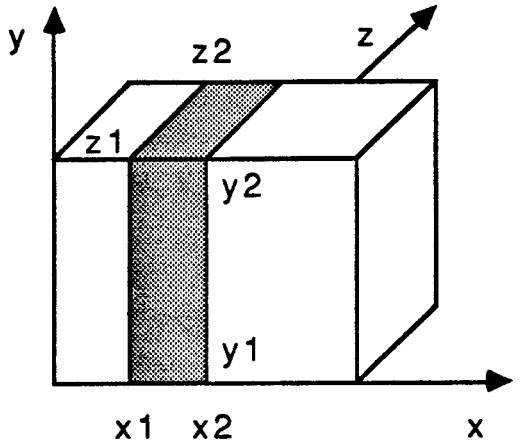
Du fait que chaque processeur objet est typé, c'est à dire que chaque processeur objet traite un type d'objet donné, le processeur objet ne peut traiter un groupement d'objets élémentaires de types différents. Donc, dans cette configuration, la scène peut se décomposer en éléments simples et en groupements d'éléments élémentaires qui sont de même type. La composition est reportée sur une couche ultérieure de traitement.

Donc, cette configuration, ne nous permet pas d'optimiser les performances (et surtout au niveau temps d'execution). Au contraire, d'autres problèmes apparaissent comme le problème d'équilibrage de charges et l'impossibilité de traiter des groupements d'objets élémentaires de types différents.

2 - PARTAGE SUIVANT LES ZONES D'OBJETS:

Tout d'abord, essayons de préciser qu'est ce qu'une zône ? Une zône peut être définie dans deux espaces:

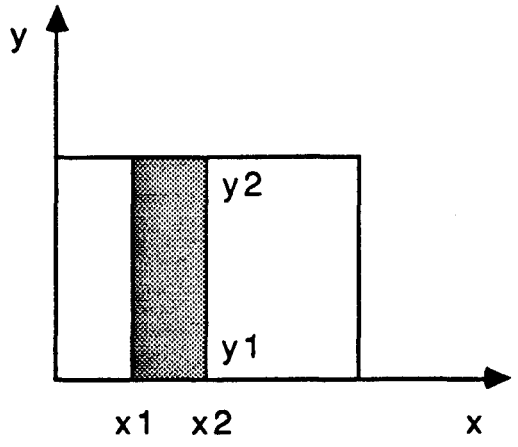
* Soit dans un espace 3D qui est l'espace de création.



une zône définie dans un espace 3D

Fig 4

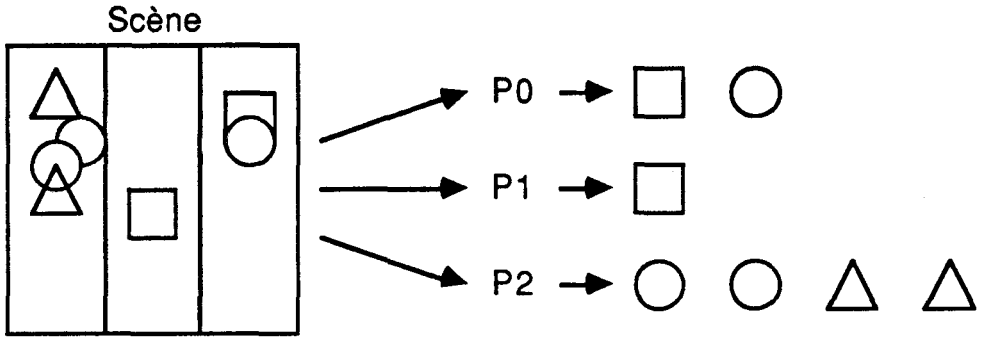
* Soit dans un espace 2D qui est la projection de l'espace de création.



une zône définie dans un espace 2D

Fig 5

Dans cette configuration, on va affecter à chaque processeur objet une zône de la scène. Tous les objets appartenant à la même zône, seront traités par le même processeur objet.



Partage suivant les zônes d'objets

Fig 6

Parmi les problèmes qui apparaissent en étudiant cette configuration, on trouve:

a - Problème d'équilibrage de charges:

Du fait que chaque processeur objet s'occupe d'une partie de la scène, le taux d'utilisation d'un processeur objet donné dépend directement de la scène. Donc, si par exemple, on se trouve en présence d'une scène contenue dans une seule zône, notre configuration parallèle se comportera comme une configuration séquentielle constituée que d'un seul processeur. Dans cette configuration, dans la plus part des cas, il y aura certains processeurs objets qui vont travailler plus que d'autres. Donc, cette configuration, induit également un problème d'équilibrage de charges.

b - Vitesse:

Du fait du non équilibrage de charges entre les processeurs objets, le temps d'execution d'une scène donnée dépend du processeur objet qui s'occupe de la zône qui renferme le plus d'objets élémentaires. Et comme, dans la plus part des cas, il y a certains processeurs objets qui travaillent plus que d'autres, cette configuration n'est pas la plus performante au niveau temps d'éxecution. Prenant l'exemple du cas particulier où la scène entière est contenue dans une seule zône, le temps d'execution de cette scène, dans ce cas là, est celui d'une configuration séquentielle composée d'un seul processeur (puisque, dans ce cas là, il n'y a qu'un seul processeur objet qui va travailler).

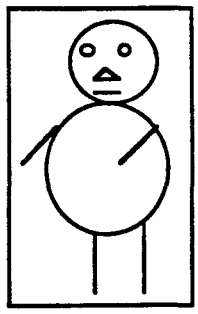
c - Problème de la localisation d'objets mobiles:

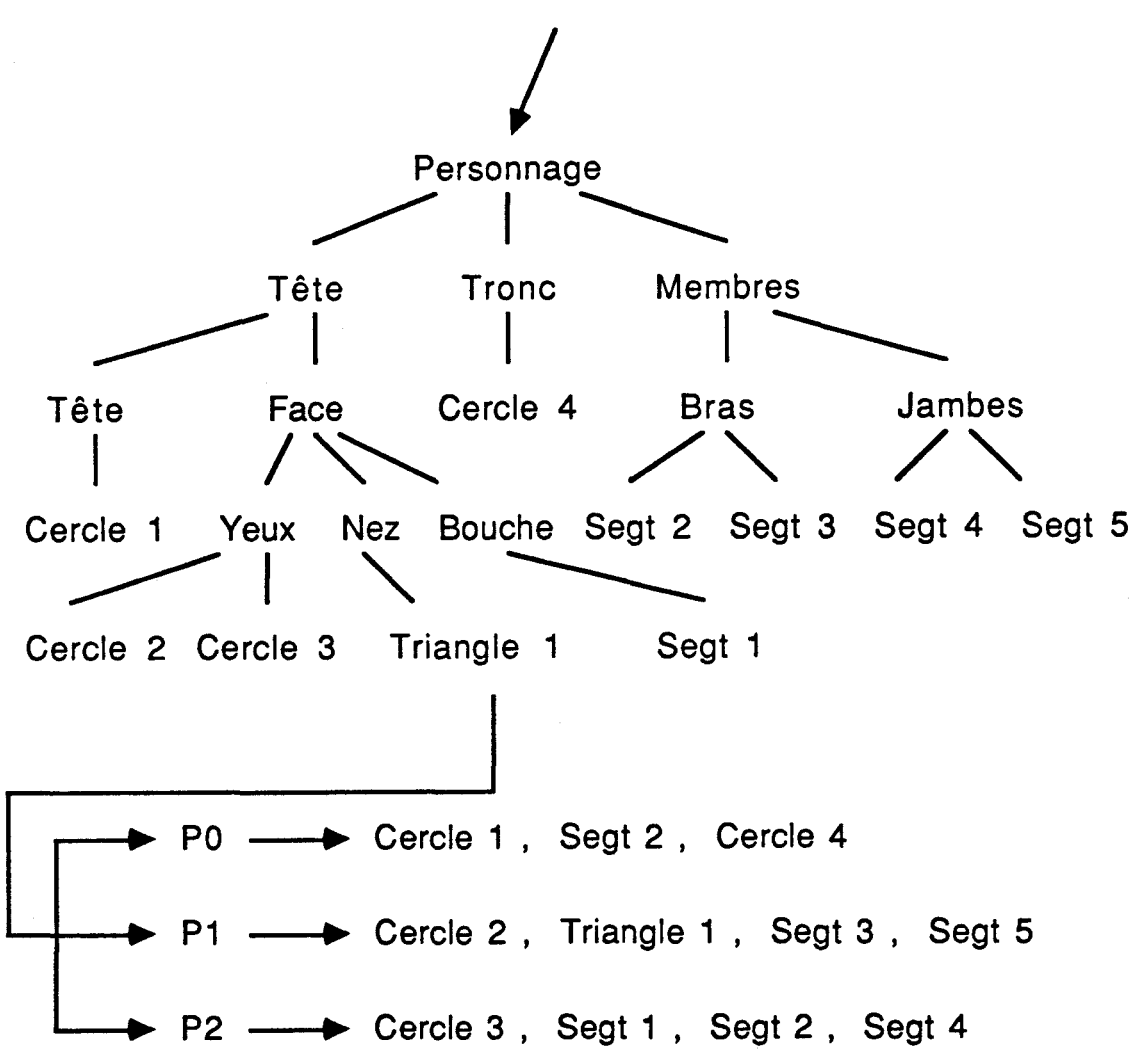
Un autre problème apparait, c'est celui de la localisation des objets mobiles (dans le cas d'une animation). Effectivement, ces objets mobiles ne peuvent être localisés, vu qu'ils sont mobiles continuellement (images mobiles). Donc, cà sera difficile de détérminer la zône à laquelle ils appartiennent et par conséquent le processeur objet qui devrait s'en occuper.

Donc, cette configuration, ne nous permet pas d'optimiser les performances (et surtout au niveau temps d'execution). Au contraire, d'autres problèmes apparaissent comme le problème d'équilibrage de charges et le problème de la localisation des objets mobiles.

3 - PARTAGE SUIVANT L'ARBORESCENCE D'OBJETS:

Dans cette configuration, la scène sera représentée par un arbre dont les feuilles seront les objets (éléments simples: segment, cercle, triangle, rectangle). On va affecté à chaque processeur objet une feuille de l'arborescence, ce qui correspond à un objet de la scène.





Partage suivant l'arborescence d'objets

Fig 7

Dans cette configuration, pour la répartition des tâches sur les processeurs objets, on adoptera la technique du "premier libre, premier servi". Du fait de l'utilisation de cette technique, pour la répartition des tâches, les processeurs objets auront presque la même charge à condition qu'ils aient tous la même compétence. Donc, dans cette configuration, le problème d'équilibrage de charges ne se pose pas. Par conséquent, cette configuration est la plus performante (au niveau temps d'execution), vu que dans la plus part des cas, tous les processeurs objets auront le même travail à fournir. Ici, le problème du traitement des groupements, composés d'objets élémentaires de types différents, ne se pose pas, puisque les processeurs objets ne sont pas typés. Quand au problème de la localisation d'objets mobiles, il n'existe pas dans cette configuration. En effet, il n'y a pas lieu de localiser les objets élémentaires.

En résumé, l'approche suivant le type d'objets est à rejeter puisque celle-ci pose le problème d'équilibrage de charges, celui de l'impossibilité de traiter un groupement d'objets élémentaires de types différents. En plus, cette approche ne permet pas d'avoir de bonnes performances. La configuration suivant les zônes d'objets est à rejeter aussi, puisque cette dernière pose le problème d'équilibrage de charges et celui de la localisation d'objets mobiles. Et elle n'est pas la plus performante. Par contre, on retiendra l'approche suivant l'arborescence d'objets puisque celle çi nous permet d'obtenir de très bonnes performances et ne pose aucun des problèmes posés par les autres configurations.

III - ETUDE DU PARTAGE SUIVANT L'ARBORESCENCE D'OBJETS:

Le système, répondant au partage suivant l'arborescence d'objets, proposé ici, est le suivant:

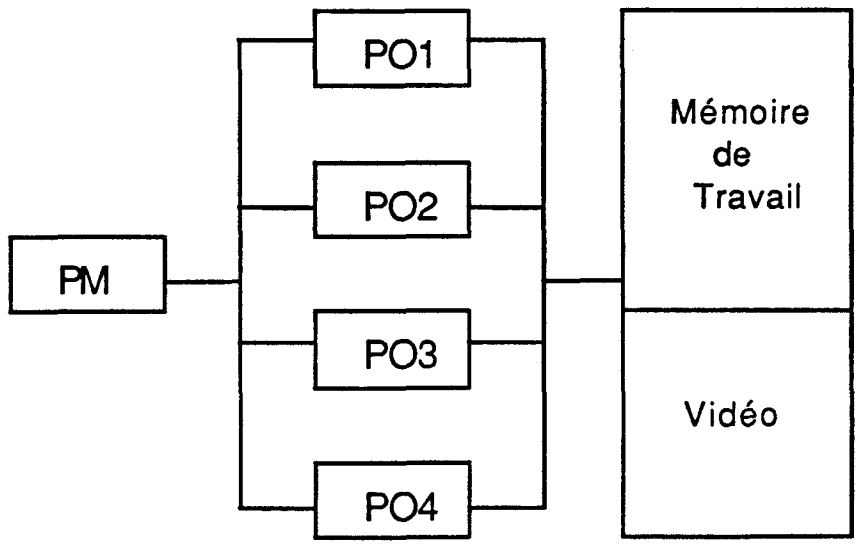


Fig 8

Il se compose d'un processeur maître, d'un certain nombre de processeurs objets, d'un processeur de cohérence, d'une mémoire de travail et d'une mémoire image.

Le problème se pose au niveau de la représentation des images en sortie des processeurs objets. On distingue trois solutions:

* Connecter directement les sorties des processeurs objets à la mémoire image. Ce qui nécessite d'avoir une mémoire à plusieurs accés. Ce qui est trés coûteux. Ceci nécessite aussi d'avoir un mécanisme de gestion de conflits (qui risquent d'être nombreux).

* Mettre des buffers à la sortie des processeurs objets pour récupérer les objets. Il faut aussi, avoir un circuit spécialisé qui videra les buffers pour mettre les données correspondantes dans la mémoire image. Cette solution est trés coûteuse. En effet, les

buffers doivent être d'une taille importante pour représenter la mémoire image. Du fait que le circuit spécialisé travaille pixel par pixel, cette solution n'offre pas de trés bonnes performances.

* Mettre une mémoire classique de travail à la sortie des processeurs, et avoir un processeur qui s'occupe du transfert de l'objet, de la mémoire de travail vers la mémoire image. Cette solution exige du processeur d'une part, de travailler par blocs de pixels, et d'autre part, d'être trés rapide. On retiendra cette solution, dans le sens où le processeur peut être le TMS 34010, puisque ce dernier est rapide et permet de travailler par blocs de pixels. Cette solution nous permet d'obtenir de bonnes performances. Nous utiliserons cette solution.

Un autre problème se pose: c'est celui de la composition d'immages (ou de la cohérence).

1 - PROBLEME DE COHERENCE (COMPOSITION):

Supposons que deux processeurs objets traitent deux objets A et B. Le problème de cohérence se pose dans le cas où ces deux objets A et B se superposent. Il se pose au moment de l'affichage.

Exemple:

Soient deux objets A et B traités respectivement par les processeurs objets P0 et P1.

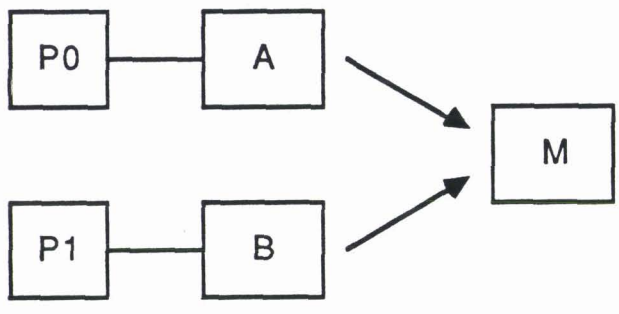


Fig 9

On suppose que ces deus objets, A et B, se superposent. On prend, par exemple, comme objet A, un rectangle, et comme objet B, un cercle.

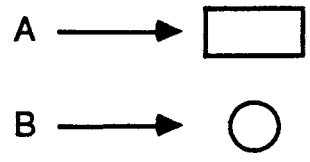


Fig 10

Le problème se pose au moment de l'affichage. On a le choix entre deux possibilités: l'objet A en premier plan, ou l'objet B en premier plan.

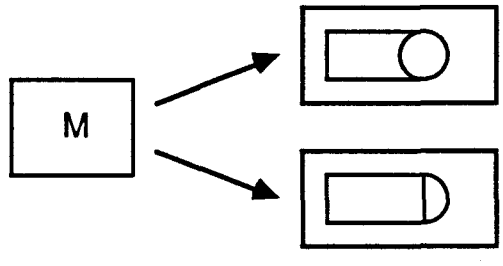


Fig 11

Pour résoudre ce problème de composition d'objets, on distingue trois solutions:

* Analyser la scène afin de trouver la liste des objets qui se superposent. Trier cette liste d'objets suivant la profondeur. Une fois le tri réalisé, on traitera cette liste d'objets séquentiellement en partant de l'objet le plus profond. Du fait de l'utilisation du traitement séquentiel, cette solution ne permet pas d'avoir de bonnes performances.

* Dans cette deuxième solution, au lieu de traiter la liste d'objets, qui se superposent, séquentiellement, on va la traiter en parallèle, tout en leur appliquant, avant de les ranger dans la

mémoire, un algorithme de z-buffer. Ce z-buffer travaillera pixel par pixel. Suite à ceci, la solution ne donne pas de bonnes performances.

* La troisième solution consiste à traiter la liste d'objets (triés suivant la profondeur), qui se superposent, en parallèle et charger un autre processeur (appelé processeur de cohérence) de leur rangement en mémoire. Cette solution exige de ce processeur d'être rapide et de travailler par blocs de pixels. On peut d'ailleurs, considérer ce processeur comme contitué de deux ADM, l'un en lecture, l'autre en écriture, plus un interpreteur de commandes. Cette solution nous permet d'obtenir de bonnes performances. Nous utiliserons cette solution.

Donc, pour résoudre le problème, où plusieurs objets sont superposés, le processeur maître devrait analyser la scène afin de la trier suivant la profondeur. Ce qui permettra d'optimiser où de garantir la cohérence. Une fois le tri réalisé, il demandera au processeur de cohérence de tranférer les objets mais suivant l'ordre déterminé auparavant (ordre suivant la profondeur).

Exemple:

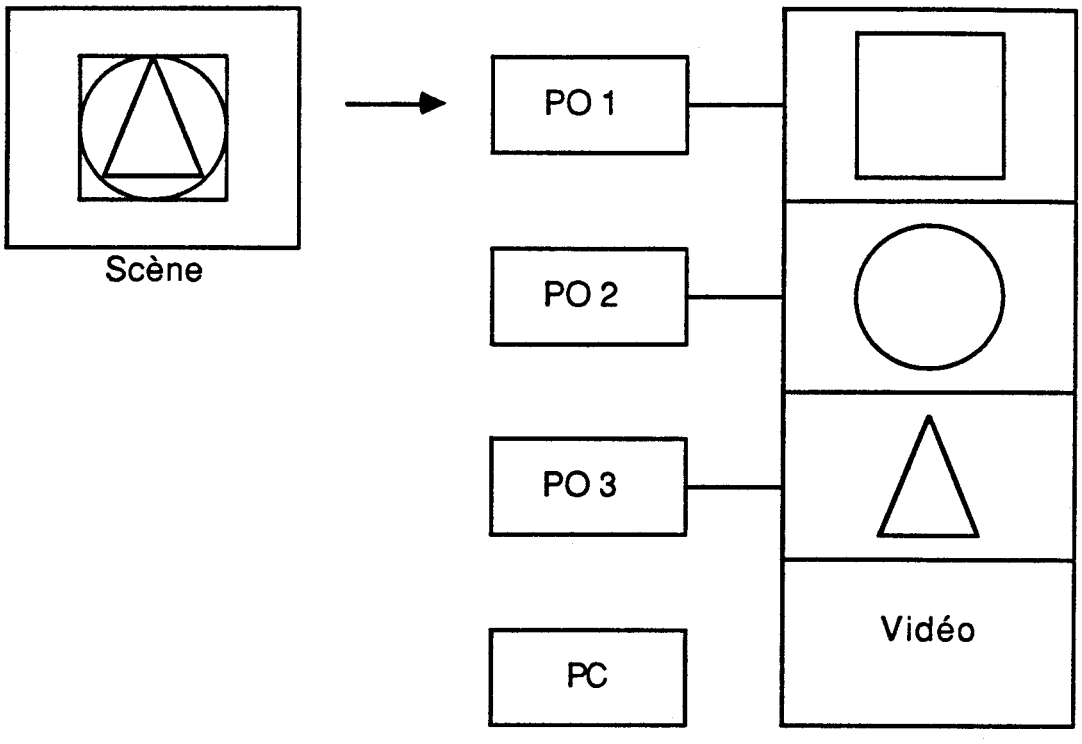


Fig 12

Le processeur maître demandera, au processeur de cohérence, de transférer d'abord le bloc 1, puis le bloc 2, et seulement après le bloc 3.

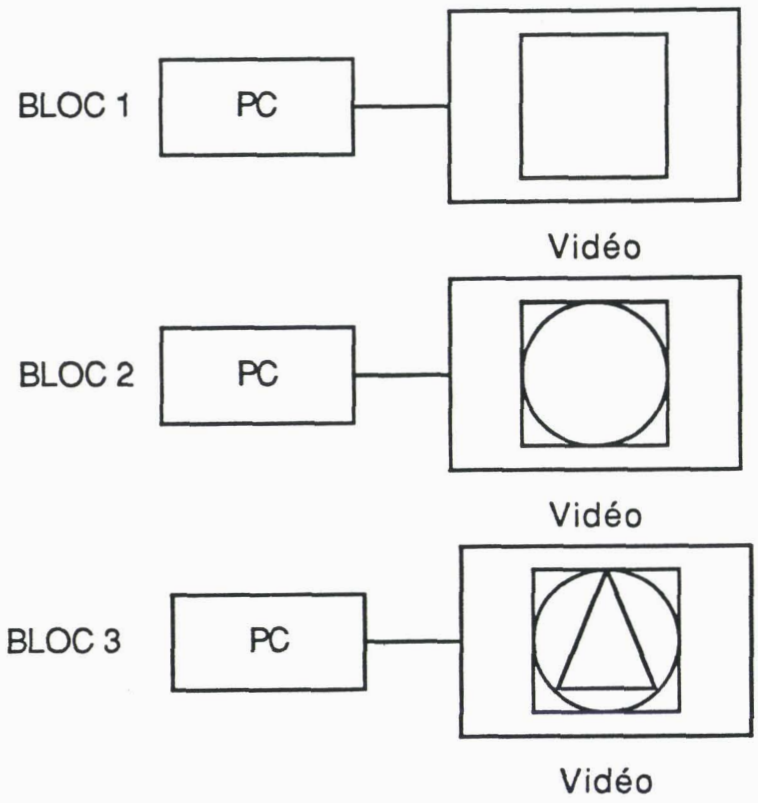


Fig 13

Le problème n'est pas simple pour les objets composés.

Exemple:

Soient les deux objets composés, A et B, suivants:

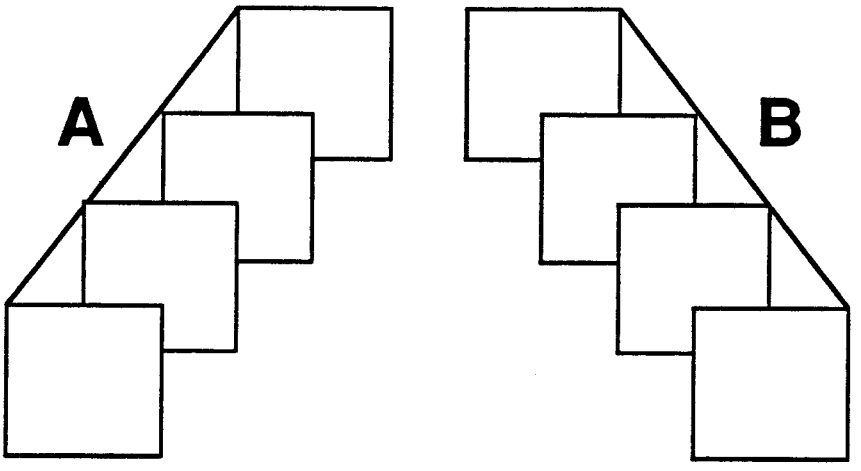


Fig 14

Le problème se pose au moment de la détermination de la position de l'un par rapport à l'autre (suivant la profondeur):

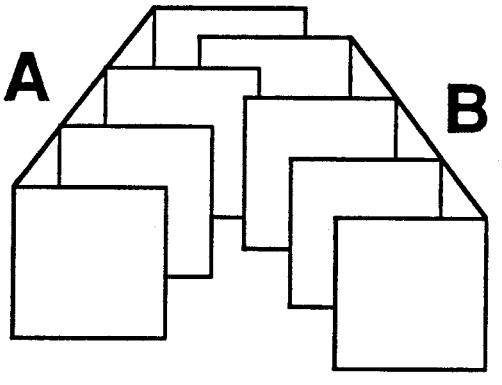


Fig 15

On résoudra ce problème en décomposant ces objets composés en éléments simples.

Exemple:

Pour l'objet A, on obtient:

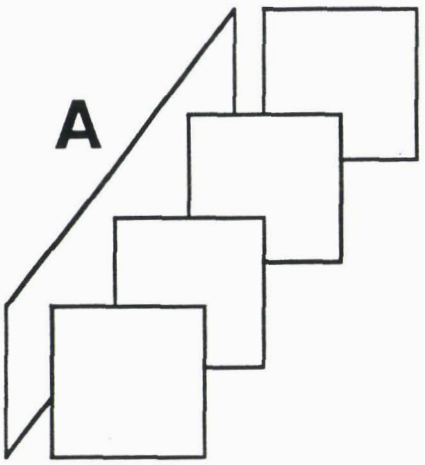


Fig 16

On peut avoir affaire aussi, à des objets définis avec un attribut logique (ou, et, ou exclusif, max, min, ... etc), arithmétique (+, -, ...), ou voir même un attribut de transparence. Ceci exige du processeur de cohérence de posséder des opérations travaillant, à une vitesse importante, sur des blocs de pixels et avec des attributs logique, arithmétique, ou de transparence, ... etc. Ces opérations ne sont offertes que par des processeurs graphiques spécialisés. Ce qui veut dire que le processeur de cohérence doit être un processeur graphique spécialisé. Donc, en utilisant un TMS 34010 comme processeur de cohérence, on aura résolu ce problème. En effet, l'opération, de transfert de blocs de pixels, est une opération qui est offerte (avec les opérateurs logiques, arithmétiques, transparence, ... etc), à trés grande vitesse, par le TMS 34010. Ceci justifie le choix de cette solution, puisqu'elle permet d'utiliser toutes les caractéristiques et performances du TMS 34010.

Donc le schéma global de cette configuration suivant l'arborescence d'objets est le suivant:

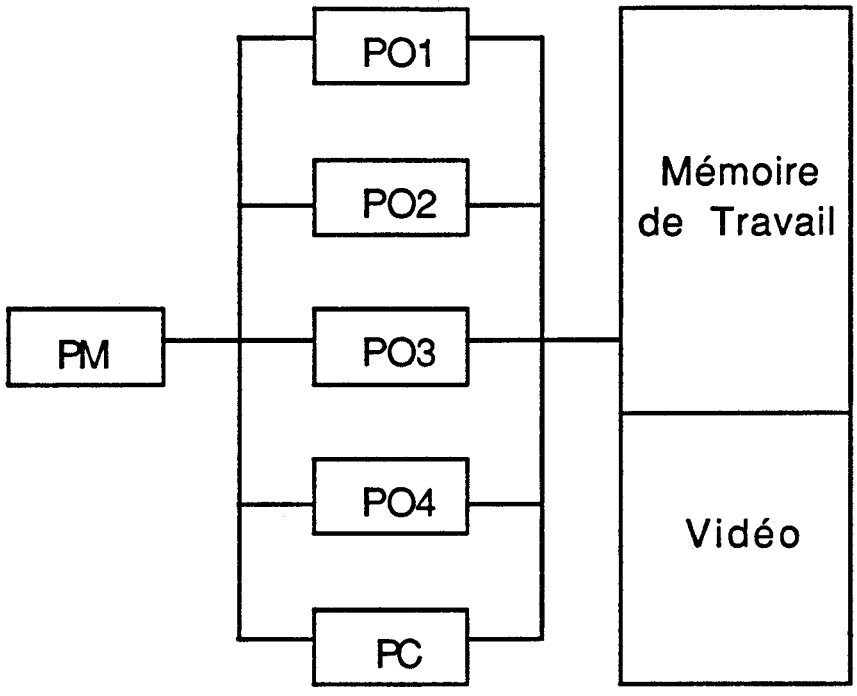


Fig 17

Cette configuration se compose: d'un processeur maître PM, d'un certain nombre de processeurs objets POi, d'un processeur de cohérence PC, d'une mémoire de travail et d'une mémoire image.

2 - PROCESSEUR DE COHERENCE:

C'est le processeur qui s'occupe du transfert d'un objet de la mémoire de travail vers la mémoire image. Dés qu'il reçoit, du processeur maître, l'ordre d'un transfert d'un objet, il envoie, au processeur maître, un accusé de réception et seulement après qu'il commence son transfert. Le problème de la cohérence est résolu par ce processeur dans le sens ou c'est lui qui traite les attributs logiques, arithmétiques, de transparence. Mais, pour ce qui est des objets superposés, c'est le processeur maître qui s'occupe, après analyse de la scène, du tri des objets suivant la profondeur, et qui envoie, au processeur de cohérence, leur demande de transfert suivant cet ordre là. Un cas particulier est celui où la scène

comporte beaucoup d'objets identiques et semblables. Dans ce cas, le processeur maître se contentera d'en traiter un seul, et en dupliquera autant de fois qu'il y en a dans la scène. Donc, il n'y aura qu'un seul objet de la série qui sera traité par un processeur objet. Et au moment du transfert, on demandera au processeur de cohérence de transférer ce bloc autant de fois qu'il y en a dans la scène, et aux positions voulues de la scène. Donc, grâce au processeur de cohérence, on peut profiter du phénomène de la duplication qui nous permet de gagner beaucoup de temps.

Exemple:

Suite à certaines commandes explicites de duplication, le processeur maître, constate qu'il a affaire au même objet. Donc, il demande à un processeur objet de faire le traitement de cet objet:

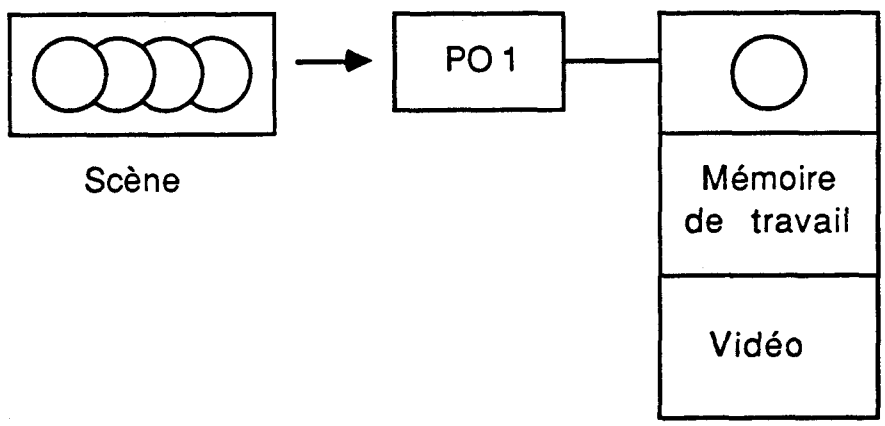


Fig 18

Après, il demande, au processeur de cohérence, de transférer 4 fois le même bloc, aux positions prises sur la scène:

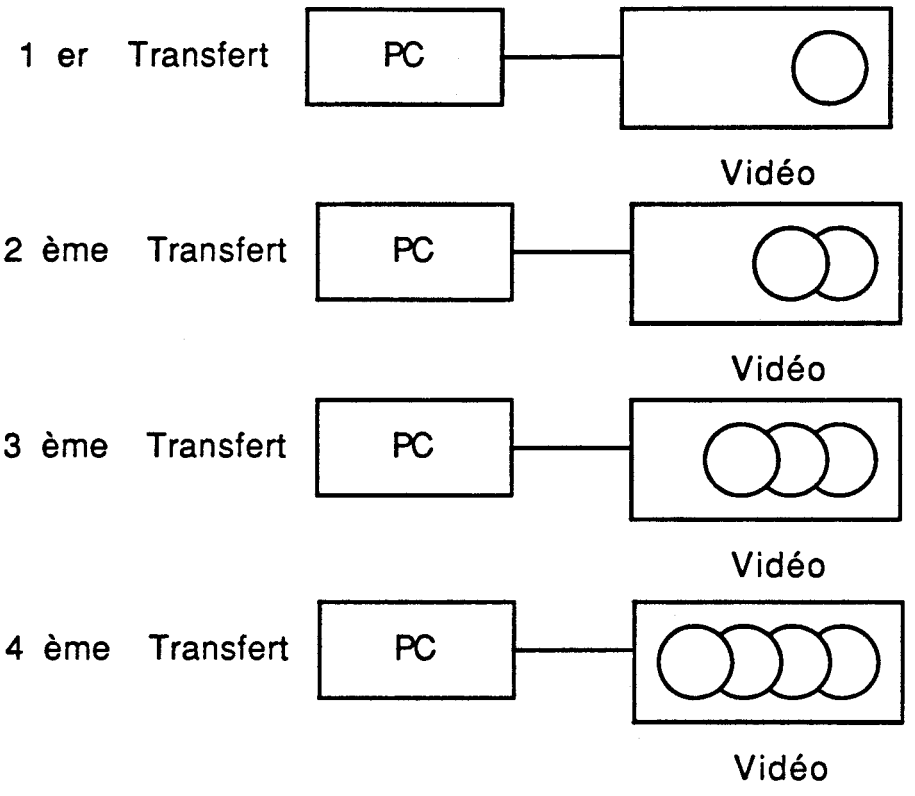


Fig 19

Après l'exécution du transfert, le processeur de cohérence envoie un message, au processeur maître, lui indiquant la fin du transfert et qu'il est prêt à exécuter un autre transfert. Mais, pour accélérer d'avantage le processus de transfert, il est préférable de faire appel à une file (une FIFO) qui va contenir tous les transferts à executer. Cette file sera intercalée entre le processeur maître et le processeur de cohérence. Elle sera alimentée par le processeur maître.

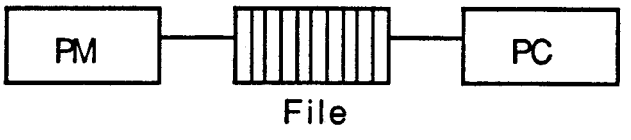


Fig 20

Donc, dés que le processeur de cohérence termine son transfert, il consulte la file. Si, la file n'est pas vide, il chertchera le prochain transfert à faire et commence à l'executer. Il peut aussi, comme les processeurs objets, traiter une tâche représentant un objet ou une partie de la scène. En fait, on peut représenter, l'état du processeur de cohérence, par un automate fini à 2 états: état occupé, état innocupé.

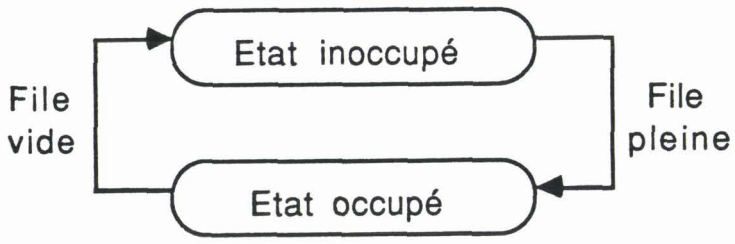


Fig 21

3 - FONCTIONNEMENT DU PROCESSEUR DE COHERENCE:

Dés que le processeur maître reçoit un message d'un processeur objet lui indiquant la fin de traitement d'un objet, le processeur maître dépose un message, dans la file, au processeur de cohérence dans lequel il lui demande d'accomplir la tâche du transfert de l'objet de la mémoire de travail vers la mémoire image avec éventuellement un traitement d'attribut logique, arithmétique, où de transparence à faire. Le processeur de cohérence, en consultant la file, trouve le message, après quoi il commence le transfert demandé. Une fois que le processeur de cohérence, termine son transfert, il ira consulter la file à la recherche d'un autre transfert à executer.

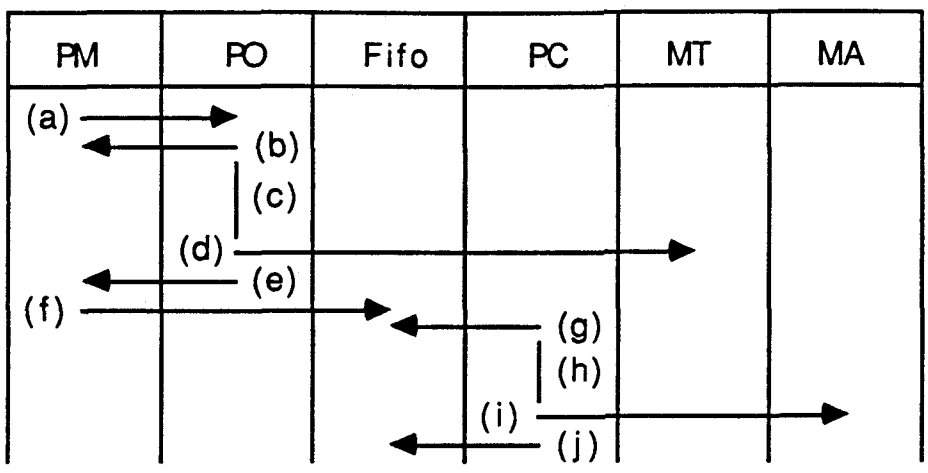


Fig 22

PM : processeur maître
PO : processeur objet
Fifo : file de transfert
PC : processeur de cohérence
MT : mémoire de travail
MA : mémoire d'affichage
(a) : demande de traitement d'un objet
(b) : envoi d'accusé de reception
(c) : traitement de l'objet
(d) : objet produit dans la mémoire de travail
(e) : fin de traitement de l'objet
(f) : demande de transfert de l'objet
(g) : consultation de la file de transfert
(h) : transfert de l'objet
(i) : objet transféré dans la mémoire d'affichage
(j) : consultation de la file de transfert

4 - PROCESSEUR MAITRE:

Mis à part des tâches classiques (gestion E/S, gestion d'interruptions, gestion de la mémoire, ...), il s'occupe de l'analyse et du découpage de la scène en éléments simples (triangle, rectangle, cercle, segment, ...), et de leur distribution sur les processeurs objets. Il peut aussi, comme les processeurs objets, traiter une tâche représentant un objet ou une partie de la scène. Il s'occupe aussi, de la recherche des objets de la scène, qui se superposent, et de leur tri suivant la profondeur.

Pour la décomposition d'objets complexes en éléments simples, plusieurs algorithmes ont été developpés. La raison de l'existence de tels algorithmes est la facilité des traitements appliqués à l'objet. En effet, les traitements appliqués à l'objet tels que translation, rotation, remplissage, ..., sont plus faciles en utilisant les éléments simples résultants qu'en utilisant l'objet lui-même.

Exemple:

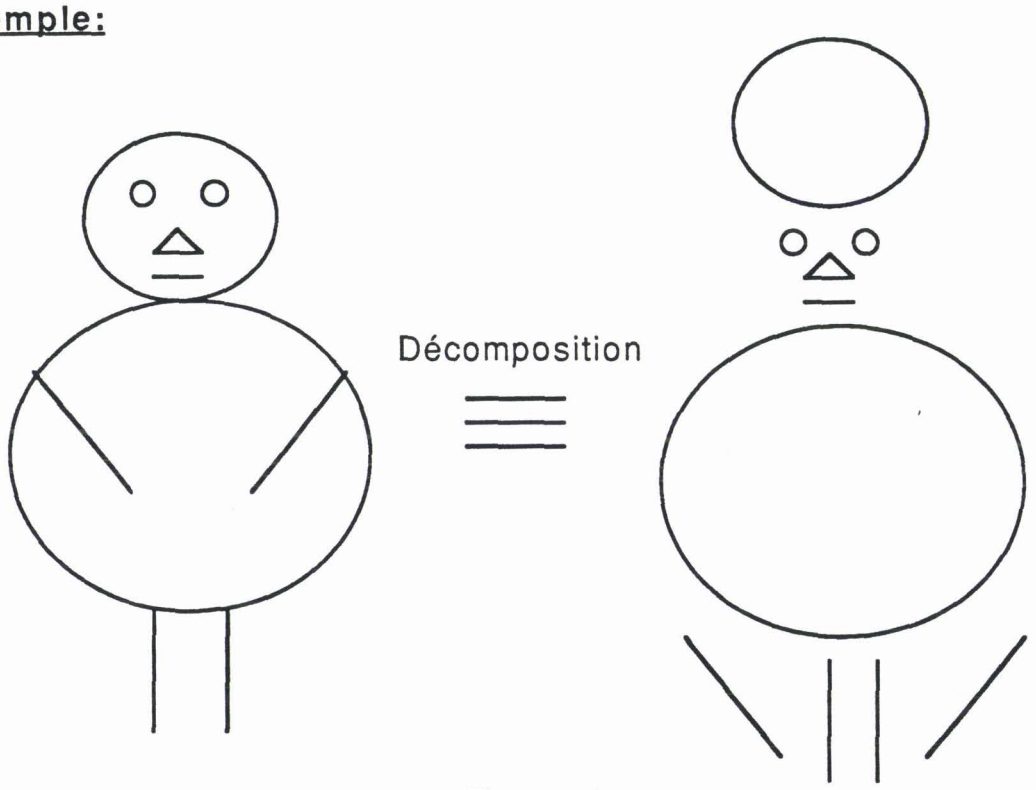


Fig 23

5 - PROCESSEUR OBJET:

C'est le processeur qui s'occupe de l'exécution de la sous-tâche. Dés qu'il reçoit, du processeur maître, la sous-tâche à traiter, il envoie, au processeur maître, un accusé de réception et seulement après qu'il commence à traiter la sous-tâche. Le problème de la cohérence n'est traité qu'au niveau du processeur de cohérence. Après l'exécution de la sous-tâche, le processeur objet envoit un message au processeur maître lui indiquant la fin d'occupation du processeur objet. En fait, là aussi, on peut représenter l'état du processeur objet par un automate fini à 2 états: état occupé, état inoccupé.

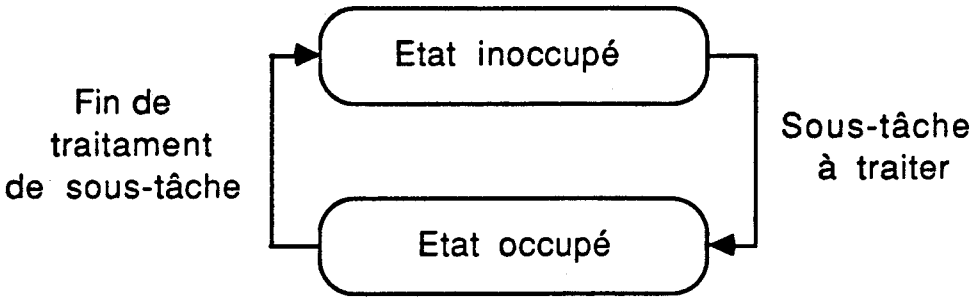


Fig 24

Chaque processeur objet est identifié par un numéro unique c'est à dire qu'on ne peut pas trouver deux processeurs objets identifiés par le même numéro.

6 - REPRESENTATION DES OBJETS:

Pour la représentation des objets, on adoptera la structure de l'arbre. Donc, un objet complexe sera représenté par un arbre, dont les feuilles seront les éléments simples (segment, cercle, rectangle, triangle).

Exemple:

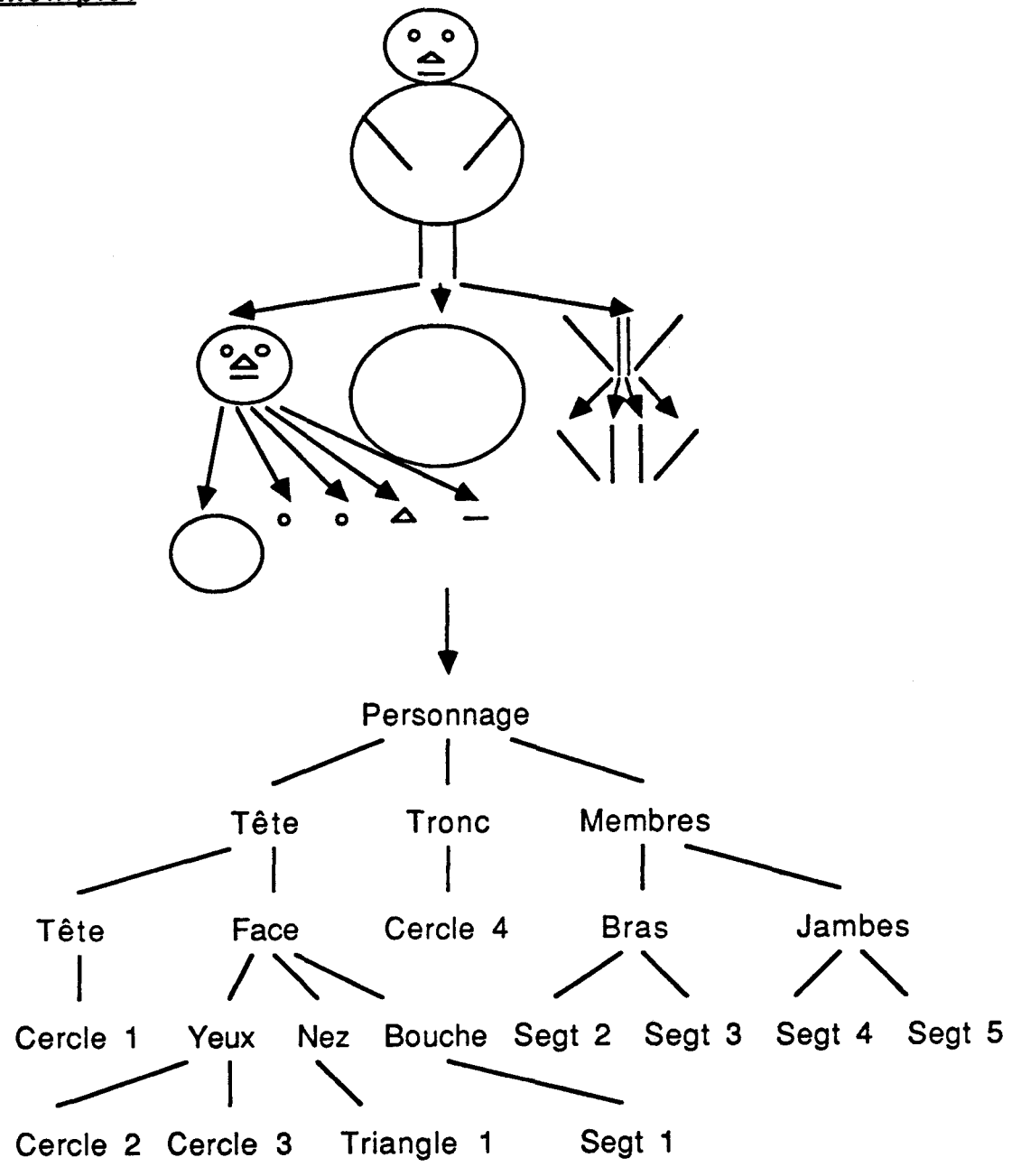


Fig 25

Une éventuelle implémentation matérielle de cette représentation des objets nous imposera de:

* figer le nombre de niveaux de l'arborescence
* mettre toutes les feuilles au niveau le plus bas

7 - Répartition des tâches élémentaires:

Pour la répartition des traitements des tâches élémentaires sur les processeurs objets, on adoptera la technique du "premier libre premier servi": il faudrait savoir lesquels d'entre eux sont au repos. C'est pourquoi nous allons mémoriser tous les travaux en cours et les numéros des processeurs objets correspondants dans une table en mémoire du processeur maître. Cette table sera scrutée à chaque nouvelle répartition de traitements, par le processeur maître.

Exemple:

Travaux en cours	N° PO
Cercle 1	0
Cercle 2	1
Cercle 3	2
Triangle 1	3

Travaux en attente
Segment 1
Cercle 4
Segment 2
Segment 3
Segment 4
Segment 5

Fig 26

Lors d'une répartition donnée de traitements, le processeur maître balaye la table des travaux en cours afin de voir s'il y a un processeur objet de libre. Deux cas peuvent se présenter:

1er CAS:

Le processeur maître trouve un processeur objet libre. A ce moment, il envoit un message au processeur objet dans lequel il lui demande de faire le traitement d'une tâche élémentaire donnée. Le processeur objet, une fois qu'il reçoit le message, il envoie un accusé de réception au processeur maître, et commence son traitement. Le processeur maître, dés qu'il reçoit l'accusé de réception, ajoute dans la table des travaux en cours, le dernier travail alloué avec le numéro du processeur objet qui s'en occupe.

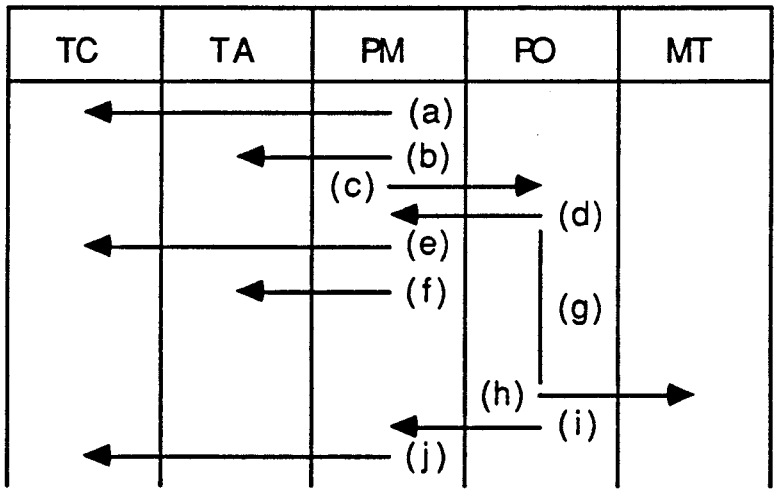


Fig 27

TC : table des travaux en cours
TA : table des travaux en attente
PM : processeur maître
PO : processeur objet
MT : mémoire de travail
(a) : recherche d'un processeur objet libre
(b) : recherche du prochain objet à traiter
(c) : demande de traitement de l'objet
(d) : envoi d'accusé de reception
(e) : remise à jour de la table des travaux en cours
(f) : remise à jour de la table des travaux en attente
(g) : traitement de l'objet
(h) : objet produit (adr) dans la mémoire de travail
(i) : fin de traitement de l'objet
(j) : remise à jour de la table des travaux en cours

Exemple:

Supposons qu'on soit dans la situation suivante:

Travaux en cours	N° PO
Cercle 1	0
Cercle 3	2
Triangle 1	3

Travaux en attente
Segment 1
Cercle 4
Segment 2
Segment 3
Segment 4
Segment 5

Fig 28

Après scrutation de la table, des travaux en cours, par le processeur maître, on trouve le processeur objet n° 1 qui est libre, et auquel on allouera le prochain travail:

Travaux en cours	N° PO
Cercle 1	0
Cercle 3	2
Triangle 1	3
Segment 1	1

Travaux en attente
Cercle 4
Segment 2
Segment 3
Segment 4
Segment 5

Fig 29

2ème CAS:

Aucun processeur objet n'est libre. A ce moment, le processeur maître doit attendre jusqu'à ce qu'il reçoit un message de fin de traitement d'une tâche allouée à un processeur objet. Une fois que le processeur maître reçoit le message de fin de traitement, il supprime le travail, qui vient de se terminer, et le numéro du processeur correspondant de la table des travaux en cours. Après quoi, nous nous retrouvons dans le 1er cas (cas où il y a un processeur objet libre). Donc, à partir de là, le processeur maître n'a qu'à faire le même travail que dans le premier cas.

Exemple:

Soit la situation suivante:

Travaux en cours	N° PO
Cercle 1	0
Cercle 2	1
Cercle 3	2
Triangle 1	3

Fig 30

Supposons que le processeur objet n° 1 a terminé sa tâche. Il envoit un message, au processeur maître, pour le prévenir de la fin du traitement. Dés que, le processeur maître reçoit le message, il remet la table à jour:

Travaux en cours	N° PO
Cercle 1	0
Cercle 3	2
Triangle 1	3

Fig 31

8 - ALLOCATION DE LA MEMOIRE DE TRAVAIL:

C'est le processeur maître qui s'occupe de l'allocation de la mémoire de travail aux processeurs objets. Avant d'allouer le traitement d'un objet donné au processeur objet, le processeur maître calcule le nombre de pages nécessaires à ce traitement en calculant le rectangle dans lequel l'objet est inscrit. Au moment, de l'allocation de ce traitement à un processeur objet, on lui alloue aussi le nombre de pages de mémoire de travail nécessaires pour ce traitement.

Exemple:

Supposons que le prochain objet à traiter soit un triangle. Le processeur maître va essayer de trouver le rectangle, dans lequel le triangle est inscrit:

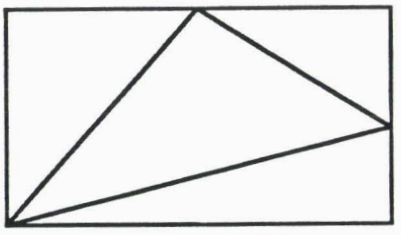


Fig 32

A partir de là, le processeur maître peut calculer le nombre de pixels, et par conséquent, la place mémoire équivalente nécessaire:

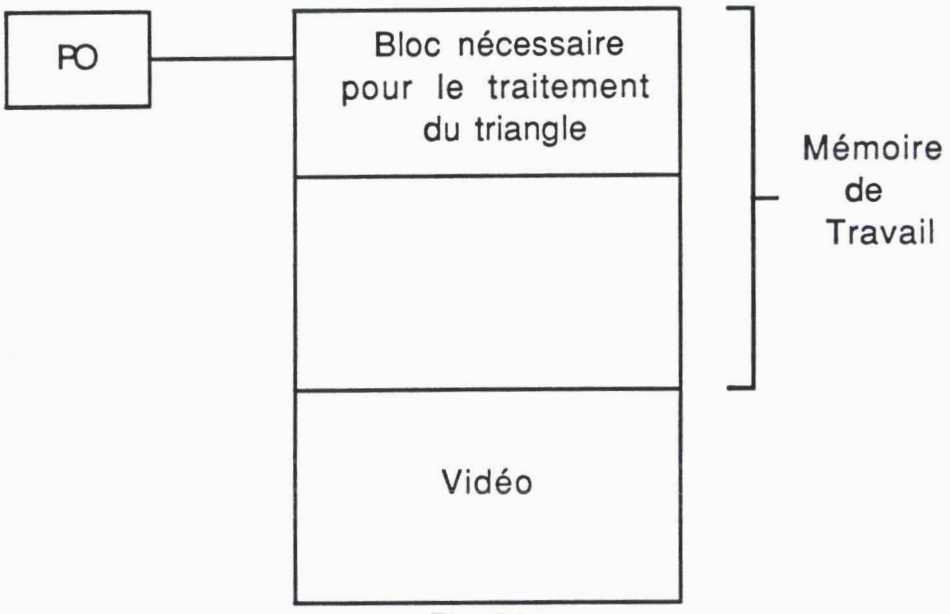


Fig 33

A la fin de traitement d'une tâche, le processeur maître restitue la place mémoire de travail allouée. On remarque qu'il est nécessaire d'avoir une gestion d'espaces libres de la mémoire de travail.

9 - COMMUNICATION:

On distingue deux communications: la communication processeur maître-processeur objet et celle du processeur maître-processeur de cohérence (file). Pour la première communication: puisque les processeurs objets sont complètement identifiés par des numéros, le processeur maître utilisera ces numéros pour communiquer avec les processeurs objets.

La communication processeur objet -> processeur maître, consiste soit à prévenir le processeur maître de la fin des travaux qui lui a été alloués, soit à envoyer, au processeur maître, l'accusé de réception du message. Tandis que les communications processeur maître -> processeur de cohérence (file) et processeur maître -> processeur objet consistent à allouer aux processeur objet et processeur de cohérence des tâches à effectuer.

IV - EVALUATION:

On va faire cette évaluation en prenant l'exemple d'une tâche triangulaire.

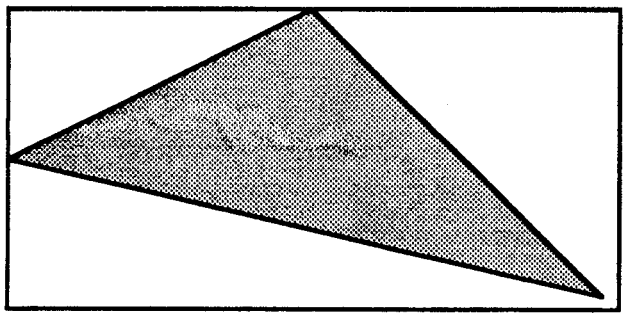


Fig 34

Le triangle qui sera utilisé dans cette évaluation a, pour extremités, les trois couples (x, y) de coordonnées suivants: (0,240), (0, 320) et (630, 470).

* Essayons de calculer le temps nécessaire pour qu'un TMS 34010 trace 1000 triangles remplis (triangle dont la taille est celle décrite ci-dessus).

Après simulation, on constate que le TMS 34010 met un temps d'environ 12 min 55 sec pour tracer ces 1000 triangles.

* Maintenant, essayons de trouver le temps nécessaire pour que quatre TMS 34010 traite ces 1000 triangles en parallèle. Donc, çà reviendrait à calculer le temps nécessaire pour qu'un TMS 34010 trace 250 triangles remplis.

Après simulation, on remarque que le TMS 34010 met un temps d'environ 3 min 14 sec pour tracer ces 250 triangles. On remarque que ce temps est le quart du temps mis pour le tracage des 1000 triangles par un seul processeur.

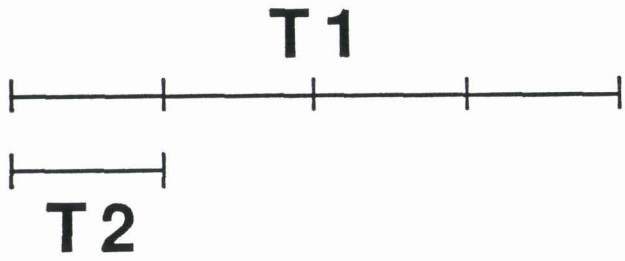


Fig 35

T1 : temps nécessaire au TMS 34010 pour tracer 1000 triangles
T2 : temps nécessaire au TMS 34010 pour tracer 250 triangles

Ce qui est tout à fait normal, dans le sens où chaque processeur trace, en parallèle, 250 triangles qui est le quart des 1000 triangles tracés par un seul processeur.

L'opération de translation se fait sur un pavé (rectangle) de pixels. Donc, pour que le processeur de cohérence translate ces 1000 triangles de la mémoire de travail vers la mémoire image, il faudrait trouver le rectangle dans lequel est inscrit le triangle (le rectangle qui encadrerait le triangle). Et pour çà, on qu'à chercher parmi les trois couples de coordonnées des extremités de ce triangle, le Xmin, Xmax, Ymin, Ymax. Le rectangle sera obtenu, en prenant pour le point le plus haut et le plus à gauche du rectangle le couple de coordonnées (Xmin, Ymin), et pour le point le plus bas et le plus à droite du rectangle le couple de coordonnées (Xmax, Ymax). Donc, dans le cadre de cet évaluation, le processeur de cohérence va translater des rectangles définis par les deux couples (x, y) de coordonnées: (0, 0) et (630, 470).

C'est l'instruction "Pixblts" du TMS 34010 qui réalise la translation. Le temps d'execution, de cette opération, dépend de la taille du pavé de pixels à translater. Il est de:

4 x M x L + 5 + Ts unités de temps

Ts : temps nécessaire à certaines initialisations (setup time = 12)
L : nombre de lignes du pavé de pixels (=630)
M : nombre de pixels par ligne (=470)

Donc le temps de translation d'un rectangle (triangle) est de:

4 x 630 x 470 + 17 unités de temps

On peut négliger le 17 devant 4 x 630 x 470. Ce qui fait un temps de 1 184 400 unités de temps.

Donc, pour translater les 1000 triangles (rectangles), le processeur de cohérence met un temps de 1 184 400 000 unités de temps. L'unité de temps du TMS 34010 est de 160 ns. Donc le temps mis pour translater ces 1000 triangles est d'environ 2 min 59 sec,

c'est à dire environ 3 min. ET le temps nécessaire pour transférer un triangle, par le processeur de cohérence, est donc d'environ 178 msec.

Il ne faut pas oublier de calculer, le temps nécessaire pour que le premier processeur objet termine le traitement du premier triangle, après quoi, le processeur de cohérence commencera son transfert. Ce temps est d'environ 776 msec.

On remarque que le temps du transfert d'un triangle représente un peu moins que le quart de celui mis pour son traitement. Essayons d'illustrer tout çà sur le chronogramme suivant:

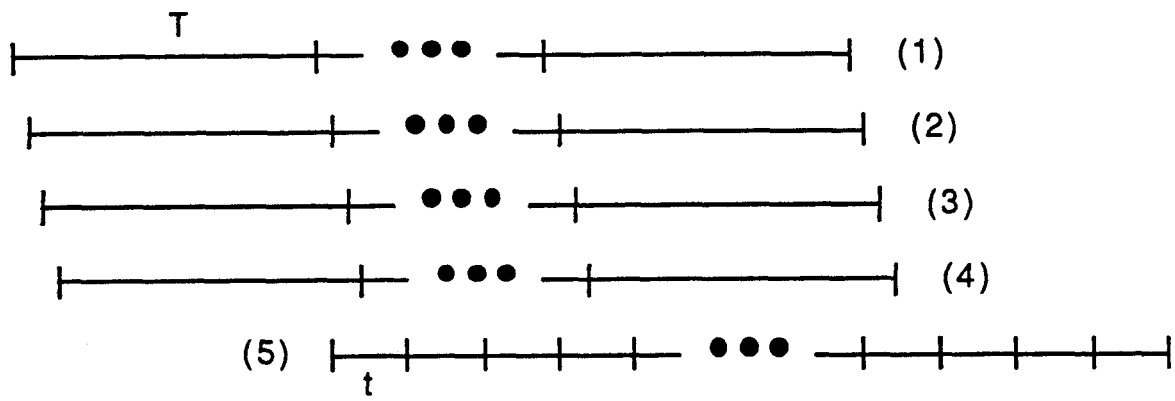


Fig 36

(1) : temps nécessaire au processeur objet 1 pour tracer 250 triangles
(2) : temps nécessaire au processeur objet 2 pour tracer 250 triangles
(3) : temps nécessaire au processeur objet 3 pour tracer 250 triangles
(4) : temps nécessaire au processeur objet 4 pour tracer 250 triangles
(5) : temps nécessaire au processeur de cohérence pour translater les 1000 triangles
T : temps nécessaire au processeur objet pour tracer 1 triangle (=776 msec)
t : temps nécessaire au processeur de cohérence pour translater 1 triangle (=178 msec)

D'après le chronogramme, le décalage entre le début de traitement, du processeur objet i, et celui du processeur objet i+1, correspond au temps nécessaire au processeur maître pour allouer la tâche au processeur objet i+1. De même, le décalage entre la fin du traitement du premier triangle du processeur objet 1, et le début de son transfert par le processeur de cohérence, correspond au temps tc nécessaire au processeur maître pour charger le processeur de cohérence de ce transfert. Le temps de transfert des quatre derniers triangles, traités en parallèle, par les quatre processeurs objets, augmenté de tc, peut être majoré par le temps nécessaire pour le traitement d'un triangle c'est à dire 776 msec. Donc le temps global mis pour le traitement en parallèle et le transfert de ces 1000 triangles est d'environ 3 min 14 sec 776 msec c'est à dire 3 min 15 sec. Donc, cette configuration parallèle suivant l'arborescence d'objets, nous permet de multiplier par quatre les performances (au niveau vitesse).

V - CONCLUSION:

On a vu que le partage objet peut se faire suivant trois configurations:

* partage suivant le type d'objets (processeurs typés)
* partage suivant les zônes d'objets
* partage suivant l'arborescence d'objets

Les deux premières solutions soulèvent quelques problèmes comme le problème d'équilibrage de charges, l'impossibilité de traiter un groupement d'objets de types différents et le problème de la localisation d'objets mobiles. Aussi, elles ne permettent pas d'obtenir de bonnes performances. Alors que la troisième solution, nous permet d'avoir de trés bons résultats tout en utilisant au mieux toutes les performances du processeur graphique spécialisé.

CONCLUSION

L'optimisation des performances d'un système graphique passe par soit la conception de circuits spécialisés soit la parallélisation. Notre étude porte a pour objectif de définir et d'étudier la parallélisation des processeurs graphiques spécialisés, en tirant avantage des deux aspects. On distingue trois grandes possibilités de parallélisme: découpage fonctionnel, découpage géométrique, et découpage objet.

1 - Le découpage fonctionnel est à rejeter, puisque celui-ci, permet le regroupement de processeurs, chacun exécutant une tâche élémentaire, alors que le processeur spécialisé offre déjà toutes ces opérations. Donc, le découpage fonctionnel ne permet pas de profiter de la puissance des processeurs graphiques spécialisés.

2 - Pour le découpage géométrique, on distingue trois solutions:

* La solution où le processeur s'occupe d'un seul pavé représentant une zône de l'écran contigue. Cette solution pose des problèmes pour les tracés offerts par le processeur graphique spécialisé, comme le tracé de segment, tracé de cercle, ... En effet, en utilisant, cette solution, on ne peut pas tirer avantage des opérations cablées offertes par le processeur graphique spécialisé et des symétries utilisées pour certains tracés. Elle pose, aussi, des difficultés pour le calcul des intersections du tracé avec les frontières des pavés. Cette solution, ne permet d'obtenir de bonnes performances.

* La solution où le processeur d'occupe de plusieurs pavés. L'ensemble de ces pavés forment une partie de l'écran non contigue. Tout ce qui a été dit auparavant, pour la solution précédente, reste valable ici. Donc, cette solution, comme la précédente, ne permet pas d'avoir de bons résultats, au niveau vitesse d'exécution.

* La dernière solution, est celle où le processeur s'occupe de plusieurs pavés, mais ces pavés sont limités à un pixel. Là aussi, l'ensemble de ces pavés forment une partie de l'écran non contigue. L'inconvénient de cette solution, c'est qu'elle ne profite pas de toute la puissance du processeur graphique spécialisé (comme les instructions cablées par exemple). Par contre, elle permet d'obtenir de bonnes performances pour toutes les tâches non traitées par le processeur graphique spécialisé (TMS 34010) comme le remplissage, l'anti-aliassage, ...

3 - Pour le découpage objet, on distingue aussi trois solutions:

* Tout d'abord, celle du partage suivant le type d'objets: cette solution soulève quelques problèmes comme le problème d'équilibrage de charges, et celui de l'impossibilité de traiter un groupement d'objets de types différents. Aussi, elle ne permet pas d'obtenir de bonnes performances.

* La solution du partage suivant les zônes d'objets: cette solution soulève, aussi, quelques problèmes comme le problème d'équilibrage de charges, et celui de la localisation d'objets mobiles. Aussi, elle ne permet pas d'avoir de bons résultats, au niveau vitesse d'exécution.

* La dernière solution, est celle du partage suivant l'arborescence d'objets: cette solution nous permet d'obtenir de trés bonnes performances tout en profitant de toute la puissance du processeur graphique spécialisé.

Il apparait, à la suite de cette étude, que le niveau de parallélisme dépend du mode de partage.

ANNEXE:

PROGRAMMES

PROGRAMME DE:

TRACE DE SEGMENT (DU TMS 34010)

```
1  #include "colors.h"
2
3  main()
4  {
5     int i,x0,y0,x1,y1;
6     long color;
7
8     init_sdb();
9     init_palet();
10    erase_screen();
11    color = 0x77777777;
12    x0=10;
13    y0=300;
14    x1=450;
15    y1=15;
16        set_color1(color);
17     for (i=1;i<=10000;i++){
18       line (x0,y0,x1,y1);
19     }
20    asm ("        TRAP 29");
21    printf ("ok !!!!\n");
22    for (;;);
23    }
```

```
1
2  *          (c) Copyright 1986, Texas Instruments, Incorporated
3  *-------------------------------------------------------------------------
4  *---------------  TMS34010 Graphics Function Library  -------------------
5  *-------------------------------------------------------------------------
6  * line function
7  *
8  *   This version of the function uses the LINE instruction, and will run
9  *   only on TMX34010A samples and above (revisions 2.0 and higher).
10 *   Another version of line is provided to run on TMX34010
11 *   (revision 1.0) samples.
12 *
13 *   Draw a line from point (xs,ys) to point (xe,ye) using Bresenham's
14 *   algorithm.
15 *
16 *   This function is designed to be called from a GSPC program.
17 *-------------------------------------------------------------------------
18 * Usage:  line(xs, ys, xe, ye);
19 *
20 * Description of the arguments:
21 *   int xs, ys;  /* starting line coordinates */
22 *   int xe, ye;  /* ending line coordinates   */
23 *
24 * Returned in register A8:  Void (undefined).
25 *
26 * Registers altered:  A8
27 *-------------------------------------------------------------------------
28 * Revision history:
29 *   12/02/86...Original version written.................Mike Asal
30 *-------------------------------------------------------------------------
31          .title   'draw a line'
32          .file          'line.asm'
33          .nolist
34          .copy          macros.hdr
35          .list
36 ;
37 ;
38 ;     DECLARE GLOBAL FUNCTION NAME
39 ;
40          .globl    _line
41 ;
42 ;
43 ;     ENTRY POINT
44 ;
45 _line:
46              MMTM    SP,B2,B7,B10,B11,B12,B13,B14
47 * Calculate XY addresses for line start and end points.
48              MOVE    A14,B14
49              MOVE    *-B14,B2,1              ; Get start x
50              MOVE    *-B14,B11,1       ; Get start y
51              SLL     16,B11
52              MOVY    B11,B2             ; B2 = (y0,x0)
53              MOVE    *-B14,B10,1       ; Get end x
54              MOVE    *-B14,B11,1       ; Get end y
55              SLL     16,B11
56              MOVY    B11,B10              ; B10 = (y1,x1)
57              MOVE    B14,A14
58 * Determine which octant line is in, and set up accordingly.
59              CLR     B7
60              SUBXY   B2,B10              ; B10 = (y1-y0,x1-x0) = (b,a)
```

```
61              JRYZ    horiz
62              JRXZ    vert
63              JRYNN   bpos
64              JRXNN   bneg_apos
65  bneg_aneg:  SUBXY   B10,B7              ; B7 = (|b|,|a|)
66              MOVI    -1,B11              ; B11 = (-1,-1)
67              JRUC    cmp_b_a
68  bneg_apos:  SUBXY   B10,B7
69              MOVX    B10,B7              ; B7 = (|b|,|a|)
70              MOVI    >FFFF0001,B11       ; B11 = (-1,1)
71              JRUC    cmp_b_a
72  bpos:           JRXNN   bpos_apos
73  bpos_aneg:  SUBXY   B10,B7
74              MOVY    B10,B7              ; B7 = (|b|,|a|)
75              MOVI    >0001FFFF,B11       ; B11 = (1,-1)
76              JRUC    cmp_b_a
77  bpos_apos:  MOVE    B10,B7              ; B7 = (|b|,|a|)
78              MOVI    >00010001,B11       ; B11 = (1,1)
79  cmp_b_a:    CLR     B12
80              MOVI    -1,B13              ; B13 = FFFFFFFF (set pattern to all 1's)
81              MOVE    B7,B0
82              SRL     16,B0               ; B0 = b
83              CLR     B10
84              MOVX    B7,B10              ; B10 = a
85              CMP     B0,B10
86              JRGT    a_ge_b
87  a_lt_b:     MOVE    B0,B10
88              MOVX    B7,B0
89              RL          16,B7           ; a and b swapped
90              MOVY    B11,B12
91              SLL     1,B0
92              SUB     B10,B0              ; B0 = 2b - a
93              ADDK    1,B10
94  * If drawing in +Y direction, use LINE 0.  Otherwise, use LINE 1.
95              MOVE    B11,B11             ; if drawing in +Y direction, use LINE 0
96              JRN     line1               ; otherwise use LINE 1
97  line0:          LINE    0
98              JRUC    done
99  a_ge_b:     MOVX    B11,B12
100             SLL     1,B0
101             SUB     B10,B0              ; B0 = 2b - a
102             MOVE    B11,B11             ; if drawing in -Y direction, use LINE 1
103             JRNN    line0               ; otherwise use LINE 0
104 line1:          LINE    1
105             JRUC    done
106 * Handle special case of horizontal line.
107 horiz:          JRXZ    pixel
108             JRXNN   do_fill
109             ADDXY   B10,B2              ; change start to (y1,x1)
110             SUBXY   B10,B7              ; make dx positive
111             MOVE    B7,B10
112 * Handle special case of vertical line.
113 vert:           JRYNN   do_fill
114             ADDXY   B10,B2              ; change start to (y1,x1)
115             NEG     B10                 ; make dy positive
116 * Draw horizontal or vertical line.
117 do_fill:    MOVE    B10,B7
118             ADDI    >10001,B7           ; no check for dx or dy overflow
119             FILL    XY
120             JRUC    done
```

```
121 * Draw dot if start and end points are the same.
122 pixel:             DRAV    B12,B2
123 * Restore registers and return.
124 done:              MMFM    SP,B2,B7,B10,B11,B12,B13,B14
125             RETS    2                       ; Return to calling routine
126             .end
```

PROGRAMME DE:

TRACE DE SEGMENT (REECRIT)

```
1  #include "colors.h"
2
3  main()
4  {
5     int i,x0,y0,x1,y1;
6     long color;
7
8     init_sdb();
9     init_palet();
10    erase_screen();
11    color = 0x77777777;
12    x0=10;
13    y0=300;
14    x1=450;
15    y1=15;
16        set_color1(color);
17     for (i=1;i<=10000;i++){
18       lines(x0,y0,x1,y1);
19     }
20    asm ("        TRAP 29");
21    printf ("ok !!!!\n");
22    for (;;);
23    }
```

```
1
2  *          (c) Copyright 1986, Texas Instruments, Incorporated
3  *----------------------------------------------------------------------
4  *---------------  TMS34010 Graphics Function Library  -----------------
5  *----------------------------------------------------------------------
6  * line function
7  *
8  *   This version of the function uses the LINE instruction, and will run
9  *   only on TMX34010A samples and above (revisions 2.0 and higher).
10 *   Another version of line is provided to run on TMX34010
11 *   (revision 1.0) samples.
12 *
13 *   Draw a line from point (xs,ys) to point (xe,ye) using Bresenham's
14 *   algorithm.
15 *
16 *   This function is designed to be called from a GSPC program.
17 *----------------------------------------------------------------------
18 * Usage:  line(xs, ys, xe, ye);
19 *
20 * Description of the arguments:
21 *   int xs, ys;  /* starting line coordinates */
22 *   int xe, ye;  /* ending line coordinates   */
23 *
24 * Returned in register A8:  Void (undefined).
25 *
26 * Registers altered:  A8
27 *----------------------------------------------------------------------
28 * Revision history:
29 *   12/02/86...Original version written.................Mike Asal
30 *----------------------------------------------------------------------
31          .title   'draw a line'
32          .file          'lines.asm'
33          .nolist
34          .copy          macros.hdr
35          .list
36 ;
37 ;
38 ;      DECLARE GLOBAL FUNCTION NAME
39 ;
40          .globl     _lines
41 ;
42 ;
43 ;      ENTRY POINT
44 ;
45 _lines:
46              MMTM     SP,A0,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12,A13
47 * Calculate XY addresses for line start and end points.
48              MOVE     *-A14,A0,1              ; Get start x
49              MOVE     *-A14,A1,1       ; Get start y
50              SLL      16,A1
51              MOVY     A1,A0             ; B2 = (y0,x0)
52              MOVE     *-A14,A1,1       ; Get end x
53              MOVE     *-A14,A2,1       ; Get end y
54              SLL      16,A2
55              MOVY     A2,A1             ; B10 = (y1,x1)
56 * Determine which octant line is in, and set up accordingly.
57              CLR      A2
58              SUBXY    A0,A1             ; B10 = (y1-y0,x1-x0) = (b,a)
59              JRZ      fin
60              JRYNN    bpos
```

```
61                JRXNN   bneg_apos
62  bneg_aneg:    SUBXY   A1,A2              ; B7 = (|b|,|a|)
63                MOVI    >-1,A3             ; B11 = (-1,-1)
64                JRUC    cmp_b_a
65  bneg_apos:    SUBXY   A1,A2
66                MOVX    A1,A2              ; B7 = (|b|,|a|)
67                MOVI    >FFFF0001,A3       ; B11 = (-1,1)
68                JRUC    cmp_b_a
69  bpos:              JRXNN   bpos_apos
70  bpos_aneg:    SUBXY   A1,A2
71                MOVY    A1,A2              ; B7 = (|b|,|a|)
72                MOVI    >0001FFFF,A3      ; B11 = (1,-1)
73                JRUC    cmp_b_a
74  bpos_apos:    MOVE    A1,A2               ; B7 = (|b|,|a|)
75                MOVI    >00010001,A3      ; B11 = (1,1)
76  cmp_b_a:      CLR     A4
77                MOVE    A2,A5
78                SRL     16,A5              ; B0 = b
79                CLR     A1
80                MOVX    A2,A1              ; B10 = a
81                CMP     A5,A1
82                JRGT    a_ge_b
83  a_lt_b:       MOVE    A5,A1
84                MOVX    A2,A5
85                RL           16,A2              ; a and b swapped
86                MOVY    A3,A4
87                SLL     1,A5
88                SUB     A1,A5              ; B0 = 2b - a
89                JRUC    label0
90  * If drawing in +Y direction, use LINE 0.  Otherwise, use LINE 1.
91  a_ge_b:       MOVX    A3,A4
92                SLL     1,A5
93                SUB     A1,A5              ; B0 = 2b - a
94  label0:       ADDK    1,A1
95                CLR     A6
96                MOVY    A2,A7
97                SRL     16,A7
98                SLL     1,A7
99                MOVX    A2,A6
100               SLL     1,A6
101               NEG     A6
102               ADD     A7,A6
103 label3:       CMPI    0,A1
104               JRLE    label10
105 label9:       CMPI    0,A10
106               JRLT    label13
107               DRAV    A3,A0
108               ADD     A6,A10
109               JRUC    label15
110 label13:      DRAV    A4,A0
111               ADD     A7,A10
112 label15:      SUBK    1,A1
113               JRNZ    label9
114               JRUC    label10
115 fin:          DRAV    A1,A0
116 label10:      MMFM    SP,A0,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12,A13
117               RETS    2
118               .end
```

PROGRAMME DE:

TRACE DE SEGMENT PARALLELE (2X2)

```
1  #include "colors.h"
2
3  main()
4  {
5     int i,x0,y0,x1,y1;
6     long color;
7
8     init_sdb();
9     init_palet();
10    erase_screen();
11    color = 0x77777777;
12    x0=10;
13    y0=300;
14    x1=450;
15    y1=300;
16        set_color1(color);
17     for (i=1;i<=10000;i++){
18       lines2(x0,y0,x1,y1,2);
19     }
20    asm ("        TRAP 29");
21    printf ("ok !!!!\n");
22    for (;;);
23    }
```

```
#include "colors.h"

main()
{
   int i,x0,y0,x1,y1;
   long color;

   init_sdb();
   init_palet();
   erase_screen();
   color = 0x77777777;
   x0=150;
   y0=450;
   x1=225;
   y1=400;
       set_color1(color);
    for (i=1;i<=100;i++){
      lines2(x0,y0,x1,y1,2);
      lines2(x0,y0,x1,y1,1);
      lines2(x0,y0,x1,y1,3);
      lines2(x0,y0,x1,y1,4);
    }
   asm ("        TRAP 29");
   printf ("ok !!!!\n");
   for (;;);
   }
```

```
1
2  *          (c) Copyright 1986, Texas Instruments, Incorporated
3  *----------------------------------------------------------------------
4  *---------------  TMS34010 Graphics Function Library  -----------------
5  *----------------------------------------------------------------------
6  * line function
7  *
8  *   This version of the function uses the LINE instruction, and will run
9  *   only on TMX34010A samples and above (revisions 2.0 and higher).
10 *   Another version of line is provided to run on TMX34010
11 *   (revision 1.0) samples.
12 *
13 *   Draw a line from point (xs,ys) to point (xe,ye) using Bresenham's
14 *   algorithm.
15 *
16 *   This function is designed to be called from a GSPC program.
17 *----------------------------------------------------------------------
18 * Usage:  line(xs, ys, xe, ye);
19 *
20 * Description of the arguments:
21 *   int xs, ys;  /* starting line coordinates */
22 *   int xe, ye;  /* ending line coordinates   */
23 *
24 * Returned in register A8:  Void (undefined).
25 *
26 * Registers altered:  A8
27 *----------------------------------------------------------------------
28 * Revision history:
29 *   12/02/86...Original version written.................Mike Asal
30 *----------------------------------------------------------------------
31          .title   'draw a line'
32          .file    'lines2.asm'
33          .nolist
34          .copy         macros.hdr
35          .list
36 ;
37 ;
38 ;     DECLARE GLOBAL FUNCTION NAME
39 ;
40          .globl    _lines2
41 ;
42 ;
43 ;     ENTRY POINT
44 ;
45 _lines2:
46             MMTM    SP,A0,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12,A13
47 * Calculate XY addresses for line start and end points.
48             MOVE    *-A14,A0,1            ; Get start x
49             MOVE    *-A14,A1,1      ; Get start y
50             SLL     16,A1
51             MOVY    A1,A0            ; B2 = (y0,x0)
52             MOVE    *-A14,A1,1      ; Get end x
53             MOVE    *-A14,A2,1      ; Get end y
54             SLL     16,A2
55             MOVY    A2,A1            ; B10 = (y1,x1)
56 * Determine which octant line is in, and set up accordingly.
57             MOVE    A1,A13
58             CLR     A2
59             SUBXY   A0,A1            ; B10 = (y1-y0,x1-x0) = (b,a)
60             JRNZ     a1
```

```
61                JRN       fin
62  a1:           JRNC      bpos
63                JRNV      bneg_apos
64  bneg_aneg:    SUBXY     A1,A2              ; B7 = (|b|,|a|)
65                MOVI      >FFFEFFFE,A3             ; B11 = (-1,-1)
66                JRUC      cmp_b_a
67  bneg_apos:    SUBXY     A1,A2
68                MOVX      A1,A2              ; B7 = (|b|,|a|)
69                MOVI      >FFFE0002,A3       ; B11 = (-1,1)
70                JRUC      cmp_b_a
71  bpos:              JRXNN     bpos_apos
72  bpos_aneg:    SUBXY     A1,A2
73                MOVY      A1,A2              ; B7 = (|b|,|a|)
74                MOVI      >0002FFFE,A3       ; B11 = (1,-1)
75                JRUC      cmp_b_a
76  bpos_apos:    MOVE      A1,A2               ; B7 = (|b|,|a|)
77                MOVI      >00020002,A3       ; B11 = (1,1)
78  cmp_b_a:      CLR       A4
79                MOVE      A2,A5
80                SRL       16,A5              ; B0 = b
81                CLR       A1
82                MOVX      A2,A1              ; B10 = a
83                CMP       A5,A1
84                JRGT      a_ge_b
85  a_lt_b:       MOVE      A5,A1
86                MOVX      A2,A5
87                RL             16,A2              ; a and b swapped
88                MOVY      A3,A4
89                SLL       1,A5
90                SUB       A1,A5              ; B0 = 2b - a
91                JRUC      label0
92  * If drawing in +Y direction, use LINE 0.  Otherwise, use LINE 1.
93  a_ge_b:       MOVX      A3,A4
94                SLL       1,A5
95                SUB       A1,A5              ; B0 = 2b - a
96  label0:       SRL       1,A1
97                ADDK      1,A1
98                CLR       A6
99                MOVY      A2,A7
100               SRL       16,A7
101               MOVE      A7,A9
102               SLL       1,A7
103               MOVX      A2,A6
104               SLL       1,A6
105               NEG       A6
106               ADD       A7,A6
107               MOVE      A6,A8
108               SRA       1,A8
109               MOVE      A5,A10
110               SRA       1,A10
111               BTST      0,A5
112               JRZ       label1
113               BTST      31,A5
114               JRZ       label1
115               SUBK      1,A10
116 label1:       CLR       A12
117               MOVE      *-A14,A2,1
118               CLR       A5
119               MOVX      A4,A5
120               MOVE      A5,A5
```

```
121                JRNZ    label2
122                SUBK    1,A2
123                JRZ     label3
124                SUBK    1,A2
125                JRNZ    label4
126                MOVE    A8,A11
127                SUB     A9,A11
128                ADD     A11,A10
129                BTST    15,A3
130                JRZ     label20
131                MOVK    1,A12
132                SLL     16,A12
133 label20:       MOVX    A3,A12
134                SRA     1,A12
135                ADDXY   A12,A0
136                JRUC    label3
137 label4:        SUBK    1,A2
138                JRNZ    label5
139                ADD     A9,A10
140                MOVY    A3,A12
141                SRA     1,A12
142                ADDXY   A12,A0
143                JRUC    label3
144 label5:        ADD     A8,A10
145                MOVE    A3,A12
146                BTST    15,A3
147                JRZ     label21
148                ORI     >00010000,A12
149 label21:       SRA     1,A12
150                ADDXY   A12,A0
151                JRUC    label3
152 label2:        SUBK    1,A2
153                JRZ     label3
154                SUBK    1,A2
155                JRNZ    label6
156                ADD     A9,A10
157                BTST    15,A3
158                JRZ     label22
159                MOVK    1,A12
160                SLL     16,A12
161 label22:       MOVX    A3,A12
162                SRA     1,A12
163                ADDXY   A12,A0
164                JRUC    label3
165 label6:        SUBK    1,A2
166                JRNZ    label5
167                MOVE    A8,A11
168                SUB     A9,A11
169                ADD     A11,A10
170                MOVY    A3,A12
171                SRA     1,A12
172                ADDXY   A12,A0
173 label3:        CMPI    0,A1
174                JRLE    label10
175 label18:       CLR     A11
176                CLR     A12
177                CLR     A5
178                MOVX    A4,A5
179                MOVE    A5,A5
180                JRZ     label17
```

```
181                BTST    15,A5
182                JRNZ    label8
183                MOVX    A0,A11
184                MOVX    A13,A12
185                CMP     A12,A11
186                JRLE    label9
187                JRUC    label10
188 label8:        MOVX    A0,A11
189                MOVX    A13,A12
190                CMP     A11,A12
191                JRLE    label9
192                JRUC    label10
193 label7:        CLR     A5
194                MOVY    A4,A5
195                MOVE    A5,A5
196                JRN     label11
197                MOVY    A0,A11
198                MOVY    A13,A12
199                CMP     A12,A11
200                JRLE    label9
201                JRUC    label10
202 label11:       MOVY    A0,A11
203                MOVY    A13,A12
204                CMP     A11,A12
205                JRLE    label9
206                JRUC    label10
207 label9:        CMPI    0,A10
208                JRLT    label13
209                CMP     A8,A10
210                JRLT    label12
211                CMP     A9,A10
212                JRGE    label12
213                DRAV    A3,A0
214                JRUC    label14
215 label12:       ADDXY   A3,A0
216 label14:       ADD     A6,A10
217                JRUC    label15
218 label13:       CMP     A8,A10
219                JRLT    label16
220                CMP     A9,A10
221                JRGE    label16
222                DRAV    A4,A0
223                JRUC    label17
224 label16:       ADDXY   A4,A0
225 label17:       ADD     A7,A10
226 label15:       SUBK    1,A1
227                JRNZ    label18
228                JRUC    label10
229 fin:           DRAV    A1,A0
230 label10:       MMFM    SP,A0,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12,A13
231                RETS    2
232                .end
```

PROGRAMME DE:

TRACE DE SEGMENT PARALLELE (4X4)

```
1  #include "colors.h"
2
3  main()
4  {
5     int i,x0,y0,x1,y1;
6     long color;
7
8     init_sdb();
9     init_palet();
10    erase_screen();
11    color = 0x77777777;
12    x0=10;
13    y0=300;
14    x1=450;
15    y1=15;
16        set_color1(color);
17     for (i=1;i<=10000;i++){
18       lines4(x0,y0,x1,y1,16);
19     }
20    asm ("        TRAP 29");
21    printf ("ok !!!!\n");
22    for (;;);
23    }
```

```
1  #include "colors.h"
2
3  main()
4  {
5     int i,x0,y0,x1,y1;
6     long color;
7     init_sdb();
8     init_palet();
9     erase_screen();
10    color = 0x77777777;
11    asm ("        TRAP 29");
12    x0=10;
13    y0=100;
14    x1=250;
15    y1=15;
16        set_color1(color);
17     for (i=1;i<=100;i++){
18      lines4(x0,y0,x1,y1,1);
19      lines4(x0,y0,x1,y1,2);
20      lines4(x0,y0,x1,y1,3);
21      lines4(x0,y0,x1,y1,4);
22      lines4(x0,y0,x1,y1,5);
23      lines4(x0,y0,x1,y1,6);
24      lines4(x0,y0,x1,y1,7);
25      lines4(x0,y0,x1,y1,8);
26      lines4(x0,y0,x1,y1,9);
27      lines4(x0,y0,x1,y1,10);
28      lines4(x0,y0,x1,y1,11);
29      lines4(x0,y0,x1,y1,12);
30      lines4(x0,y0,x1,y1,13);
31      lines4(x0,y0,x1,y1,14);
32      lines4(x0,y0,x1,y1,15);
33      lines4(x0,y0,x1,y1,16);
34     }
35    asm ("        TRAP 29");
36    printf ("ok !!!!\n");
37    for (;;);
38    }
```

```
1
2  *          (c) Copyright 1986, Texas Instruments, Incorporated
3  *---------------------------------------------------------------------
4  *---------------  TMS34010 Graphics Function Library  ----------------
5  *---------------------------------------------------------------------
6  * line function
7  *
8  *   This version of the function uses the LINE instruction, and will run
9  *   only on TMX34010A samples and above (revisions 2.0 and higher).
10 *   Another version of line is provided to run on TMX34010
11 *   (revision 1.0) samples.
12 *
13 *   Draw a line from point (xs,ys) to point (xe,ye) using Bresenham's
14 *   algorithm.
15 *
16 *   This function is designed to be called from a GSPC program.
17 *---------------------------------------------------------------------
18 * Usage:  line(xs, ys, xe, ye);
19 *
20 * Description of the arguments:
21 *   int xs, ys;  /* starting line coordinates */
22 *   int xe, ye;  /* ending line coordinates   */
23 *
24 * Returned in register A8:  Void (undefined).
25 *
26 * Registers altered:  A8
27 *---------------------------------------------------------------------
28 * Revision history:
29 *   12/02/86...Original version written.................Mike Asal
30 *---------------------------------------------------------------------
31          .title   'draw a line'
32          .file          'lines4.asm'
33          .nolist
34          .copy          macros.hdr
35          .list
36 ;
37 ;
38 ;     DECLARE GLOBAL FUNCTION NAME
39 ;
40          .globl    _lines4
41 ;
42 ;
43 ;     ENTRY POINT
44 ;
45 _lines4:
46             MMTM    SP,A0,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12,A13
47 * Calculate XY addresses for line start and end points.
48             MOVE    *-A14,A0,1              ; Get start x
49             MOVE    *-A14,A1,1      ; Get start y
50             SLL     16,A1
51             MOVY    A1,A0            ; B2 = (y0,x0)
52             MOVE    *-A14,A1,1      ; Get end x
53             MOVE    *-A14,A2,1      ; Get end y
54             SLL     16,A2
55             MOVY    A2,A1            ; B10 = (y1,x1)
56 * Determine which octant line is in, and set up accordingly.
57             MOVE    A1,A13
58             CLR     A2
59             SUBXY   A0,A1            ; B10 = (y1-y0,x1-x0) = (b,a)
60             JRZ     fin
```

```
61                JRYNN   bpos
62                JRXNN   bneg_apos
63  bneg_aneg:    SUBXY   A1,A2              ; B7 = (|b|,|a|)
64                MOVI    >FFFCFFFC,A3             ; B11 = (-1,-1)
65                JRUC    cmp_b_a
66  bneg_apos:    SUBXY   A1,A2
67                MOVX    A1,A2             ; B7 = (|b|,|a|)
68                MOVI    >FFFC0004,A3      ; B11 = (-1,1)
69                JRUC    cmp_b_a
70  bpos:              JRXNN   bpos_apos
71  bpos_aneg:    SUBXY   A1,A2
72                MOVY    A1,A2             ; B7 = (|b|,|a|)
73                MOVI    >0004FFFC,A3     ; B11 = (1,-1)
74                JRUC    cmp_b_a
75  bpos_apos:    MOVE    A1,A2              ; B7 = (|b|,|a|)
76                MOVI    >00040004,A3     ; B11 = (1,1)
77  cmp_b_a:      CLR     A4
78                MOVE    A2,A5
79                SRL     16,A5             ; B0 = b
80                CLR     A1
81                MOVX    A2,A1             ; B10 = a
82                CMP     A5,A1
83                JRGT    a_ge_b
84  a_lt_b:       MOVE    A5,A1
85                MOVX    A2,A5
86                RL           16,A2            ; a and b swapped
87                MOVY    A3,A4
88                SLL     1,A5
89                SUB     A1,A5             ; B0 = 2b - a
90                JRUC    label0
91  * If drawing in +Y direction, use LINE 0.  Otherwise, use LINE 1.
92  a_ge_b:       MOVX    A3,A4
93                SLL     1,A5
94                SUB     A1,A5             ; B0 = 2b - a
95  label0:       SRL     2,A1
96                ADDK    1,A1
97                CLR     A6
98                MOVY    A2,A7
99                SRL     16,A7
100               MOVE    A7,A9
101               SLL     1,A7
102               MOVX    A2,A6
103               SLL     1,A6
104               NEG     A6
105               ADD     A7,A6
106               MOVE    A6,A8
107               SRA     1,A8
108               SLL     1,A6
109               SLL     1,A7
110               MOVE    A5,A10
111               SRA     1,A10
112               BTST    0,A5
113               JRZ     label1
114               BTST    31,A5
115               JRZ     label1
116               SUBK    1,A10
117 label1:       CLR     A12
118               CLR     A11
119               MOVE    *-A14,A2,1
120               CLR     A5
```

```
121                MOVX    A4,A5
122                MOVE    A5,A5
123                JRNZ    label2
124                SUBK    1,A2
125                JRZ     label3
126                MOVE    A8,A5
127                SUB     A9,A5
128                SUBK    1,A2
129                JRNZ    label20
130                ADD     A5,A10
131                MOVX    A3,A12
132                SRA     2,A12
133                ADDXY   A12,A0
134                JRUC    label3
135 label20:       SUBK    1,A2
136                JRNZ    label21
137                SLL     1,A5
138                ADD     A5,A10
139                MOVX    A3,A12
140                SRA     1,A12
141                ADDXY   A12,A0
142                JRUC    label3
143 label21:       SUBK    1,A2
144                JRNZ    label22
145                ADD     A5,A10
146                SLL     1,A5
147                ADD     A5,A10
148                MOVX    A3,A12
149                SRA     1,A12
150                MOVX    A3,A11
151                SRA     2,A11
152                ADDXY   A11,A12
153                ADDXY   A12,A0
154                JRUC    label3
155 label22:       SUBK    1,A2
156                JRNZ    label23
157                ADD     A9,A10
158                MOVY    A3,A12
159                SRA     2,A12
160                ADDXY   A12,A0
161                JRUC    label3
162 label23:       SUBK    1,A2
163                JRNZ    label24
164                ADD     A8,A10
165                MOVE    A3,A12
166                SRA     2,A12
167                ADDXY   A12,A0
168                JRUC    label3
169 label24:       SUBK    1,A2
170                JRNZ    label25
171                ADD     A8,A10
172                ADD     A5,A10
173                MOVY    A3,A12
174                SRA     1,A12
175                MOVX    A3,A12
176                SRA     1,A12
177                ADDXY   A12,A0
178                JRUC    label3
179 label25:       SUBK    1,A2
180                JRNZ    label26
```

```
181                ADD     A8,A10
182                SLL     1,A5
183                ADD     A5,A10
184                MOVX    A3,A11
185                SRA     1,A11
186                MOVE    A3,A12
187                SRA     2,A12
188                ADDXY   A11,A12
189                ADDXY   A12,A0
190                JRUC    label3
191 label26:       SUBK    1,A2
192                JRNZ    label27
193                ADD     A9,A10
194                ADD     A9,A10
195                MOVY    A3,A12
196                SRA     1,A12
197                ADDXY   A12,A0
198                JRUC    label3
199 label27:       SUBK    1,A2
200                JRNZ    label28
201                ADD     A9,A10
202                ADD     A9,A10
203                ADD     A5,A10
204                MOVX    A3,A12
205                SRA     1,A12
206                MOVY    A3,A12
207                SRA     1,A12
208                ADDXY   A12,A0
209                JRUC    label3
210 label28:       SUBK    1,A2
211                JRNZ    label29
212                ADD     A8,A10
213                ADD     A8,A10
214                MOVE    A3,A12
215                SRA     1,A12
216                ADDXY   A12,A0
217                JRUC    label3
218 label29:       SUBK    1,A2
219                JRNZ    label30
220                ADD     A8,A10
221                ADD     A8,A10
222                ADD     A5,A10
223                MOVE    A3,A12
224                SRA     1,A12
225                MOVX    A3,A11
226                SRA     2,A11
227                ADDXY   A11,A12
228                ADDXY   A12,A0
229                JRUC    label3
230 label30:       SUBK    1,A2
231                JRNZ    label31
232                ADD     A9,A10
233                ADD     A9,A10
234                ADD     A9,A10
235                MOVY    A3,A12
236                SRA     1,A12
237                MOVY    A3,A11
238                SRA     2,A11
239                ADDXY    A11,A12
240                ADDXY   A12,A0
```

```
241                JRUC    label3
242 label31:       SUBK    1,A2
243                JRNZ    label32
244                ADD     A8,A10
245                ADD     A9,A10
246                ADD     A9,A10
247                MOVE    A3,A12
248                SRA     2,A12
249                MOVY    A3,A11
250                SRA     1,A11
251                ADDXY    A11,A12
252                ADDXY   A12,A0
253                JRUC    label3
254 label32:       SUBK    1,A2
255                JRNZ    label5
256                ADD     A8,A10
257                ADD     A8,A10
258                ADD     A9,A10
259                MOVE    A3,A12
260                SRA     1,A12
261                MOVY    A3,A11
262                SRA     2,A11
263                ADDXY    A11,A12
264                ADDXY   A12,A0
265                JRUC    label3
266 label5:        ADD     A8,A10
267                ADD     A8,A10
268                ADD     A8,A10
269                MOVE    A3,A12
270                SRA     1,A12
271                MOVE    A3,A11
272                SRA     2,A11
273                ADDXY   A11,A12
274                ADDXY   A12,A0
275                JRUC    label3
276 label2:        SUBK    1,A2
277                JRZ     label3
278                MOVE    A8,A5
279                SUB     A9,A5
280                SUBK    1,A2
281                JRNZ    label40
282                ADD     A9,A10
283                MOVX    A3,A12
284                SRA     2,A12
285                ADDXY   A12,A0
286                JRUC    label3
287 label40:       SUBK    1,A2
288                JRNZ    label41
289                ADD     A9,A10
290                ADD     A9,A10
291                MOVX    A3,A12
292                SRA     1,A12
293                ADDXY   A12,A0
294                JRUC    label3
295 label41:       SUBK    1,A2
296                JRNZ    label42
297                ADD     A9,A10
298                ADD     A9,A10
299                ADD     A9,A10
300                MOVX    A3,A12
```

```
301                 SRA     2,A12
302                 MOVX    A3,A11
303                 SRA     1,A11
304                 ADDXY   A11,A12
305                 ADDXY   A12,A0
306                 JRUC    label3
307 label42:        SUBK    1,A2
308                 JRNZ    label43
309                 ADD     A5,A10
310                 MOVY    A3,A12
311                 SRA     2,A12
312                 ADDXY   A12,A0
313                 JRUC    label3
314 label43:        SUBK    1,A2
315                 JRNZ    label44
316                 ADD     A8,A10
317                 MOVE    A3,A12
318                 SRA     2,A12
319                 ADDXY   A12,A0
320                 JRUC    label3
321 label44:        SUBK    1,A2
322                 JRNZ    label45
323                 ADD     A9,A10
324                 ADD     A8,A10
325                 MOVY    A3,A12
326                 SRA     1,A12
327                 MOVX    A3,A12
328                 SRA     1,A12
329                 ADDXY   A12,A0
330                 JRUC    label3
331 label45:        SUBK    1,A2
332                 JRNZ    label46
333                 ADD     A9,A10
334                 ADD     A9,A10
335                 ADD     A8,A10
336                 MOVE    A3,A12
337                 SRA     2,A12
338                 MOVX    A3,A11
339                 SRA     1,A11
340                 ADDXY   A11,A12
341                 ADDXY   A12,A0
342                 JRUC    label3
343 label46:        SUBK    1,A2
344                 JRNZ    label47
345                 SLL     1,A5
346                 ADD     A5,A10
347                 MOVY    A3,A12
348                 SRA     1,A12
349                 ADDXY   A12,A0
350                 JRUC    label3
351 label47:        SUBK    1,A2
352                 JRNZ    label48
353                 ADD     A8,A10
354                 ADD     A5,A10
355                 MOVX    A3,A12
356                 SRA     1,A12
357                 MOVY    A3,A12
358                 SRA     1,A12
359                 ADDXY   A12,A0
360                 JRUC    label3
```

```
361 label48:    SUBK    1,A2
362             JRNZ    label49
363             ADD     A8,A10
364             ADD     A8,A10
365             MOVE    A3,A12
366             SRA     1,A12
367             ADDXY   A12,A0
368             JRUC    label3
369 label49:    SUBK    1,A2
370             JRNZ    label50
371             ADD     A8,A10
372             ADD     A8,A10
373             ADD     A9,A10
374             MOVE    A3,A12
375             SRA     1,A12
376             MOVX    A3,A11
377             SRA     2,A11
378             ADDXY   A11,A12
379             ADDXY   A12,A0
380             JRUC    label3
381 label50:    SUBK    1,A2
382             JRNZ    label51
383             ADD     A5,A10
384             SLL     1,A5
385             ADD     A5,A10
386             MOVY    A3,A12
387             SRA     2,A12
388             MOVY    A3,A11
389             SRA     1,A11
390             ADDXY    A11,A12
391             ADDXY   A12,A0
392             JRUC    label3
393 label51:    SUBK    1,A2
394             JRNZ    label52
395             SLL     1,A5
396             ADD     A5,A10
397             ADD     A8,A10
398             MOVE    A3,A12
399             SRA     2,A12
400             MOVY    A3,A11
401             SRA     1,A11
402             ADDXY    A11,A12
403             ADDXY   A12,A0
404             JRUC    label3
405 label52:    SUBK    1,A2
406             JRNZ    label5
407             ADD     A9,A10
408             ADD     A8,A10
409             ADD     A8,A10
410             MOVE    A3,A12
411             SRA     1,A12
412             MOVY    A3,A11
413             SRA     2,A11
414             ADDXY    A11,A12
415             ADDXY   A12,A0
416 label3:     CMPI    0,A1
417             JRLE    label10
418 label18:    CLR     A11
419             CLR     A12
420             CLR     A5
```

```
421                MOVX    A4,A5
422                MOVE    A5,A5
423                JRZ     label7
424                JRN     label8
425                MOVX    A0,A11
426                MOVX    A13,A12
427                CMP     A12,A11
428                JRLE    label9
429                JRUC    label10
430 label8:        MOVX    A0,A11
431                MOVX    A13,A12
432                CMP     A11,A12
433                JRLE    label9
434                JRUC    label10
435 label7:        CLR     A5
436                MOVY    A4,A5
437                MOVE    A5,A5
438                JRN     label11
439                MOVY    A0,A11
440                MOVY    A13,A12
441                CMP     A12,A11
442                JRLE    label9
443                JRUC    label10
444 label11:       MOVY    A0,A11
445                MOVY    A13,A12
446                CMP     A11,A12
447                JRLE    label9
448                JRUC    label10
449 label9:        CMPI    0,A10
450                JRLT    label13
451                CMP     A8,A10
452                JRLT    label12
453                CMP     A9,A10
454                JRGE    label12
455                DRAV    A3,A0
456                JRUC    label14
457 label12:       ADDXY   A3,A0
458 label14:       ADD     A6,A10
459                JRUC    label15
460 label13:       CMP     A8,A10
461                JRLT    label16
462                CMP     A9,A10
463                JRGE    label16
464                DRAV    A4,A0
465                JRUC    label17
466 label16:       ADDXY   A4,A0
467 label17:       ADD     A7,A10
468 label15:       SUBK    1,A1
469                JRNZ    label18
470                JRUC    label10
471 fin:           DRAV    A1,A0
472 label10:       MMFM    SP,A0,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12,A13
473                RETS    2
474                .end
```

PROGRAMME DE:

TRACE DE CERCLE

```
1  #include "colors.h"
2
3  main()
4  {
5     int r,x,y,i;
6     long color;
7
8     init_sdb();
9     init_palet();
10    erase_screen();
11    color = 0x77777777;
12    x=350;
13    y=250;
14    r=200;
15        set_color1(color);
16     for (i=1;i<=100;i++){
17      circle(r,x,y);
18     }
19    asm ("        TRAP 29");
20    printf ("ok !!!!\n");
21    for (;;);
22    }
```

```
1           .width    132
2
3  *         (c) Copyright 1987, Texas Instruments, Incorporated
4  *---------------------------------------------------------------------
5  *---------------  TMS34010 Graphics Function Library  ----------------
6  *---------------------------------------------------------------------
7  * circle function
8  *
9  *   Draw outline of circle.  Radius of circle is specified by argument
10 *   r.       Center point is specified by arguments x and y.
11 *
12 *   This function is designed to be called from a GSP-C program.
13 *---------------------------------------------------------------------
14 * Usage:  circle(r, x, y);
15 *
16 * Description of stack arguments:
17 *   int r;  /* radius */
18 *   int x, y;  /* center coordinates */
19 *
20 * Returned in register A8:  Void (undefined)
21 *
22 * Registers altered:  A8
23 *---------------------------------------------------------------------
24 * Revision history:
25 *   1/9/87.....Original version written.................Jerry Van Aken
26 *---------------------------------------------------------------------
27 ;
28          .title   'circle'
29          .file         'circle.asm'
30 ;
31 ;
32 ;     DECLARE GLOBAL FUNCTION NAME
33 ;
34          .globl    _circle
35 ;
36 ;
37 ;     ENTRY POINT
38 ;
39 _circle:
40
41          MMTM         SP,A0,A1,A2,A3,A4,A5,A6,A10,A11,A12,A13,A14
42          MMTM         SP,B0,B1,B2,B7,B10,B11,B12,B13,B14
43          MOVE         *-A14,A8,1              ;get argument 'r'
44          MOVE         *-A14,A0,1              ;get argument 'x'
45          MOVE         *-A14,A1,1              ;get argument 'y'
46          TRAP     29
47 * Form XY address of circle's center point.
48          SLL         16,A1                ;concatenate x and y
49          MOVY         A1,A0                ;
50 * Get starting XY addresses for arcs in 8 octants.
51          MOVE         A8,A1                ;copy r
52          JRZ      C3                       ;jump if r = 0
53          SLL         16,A1                ;XY displacement +r::0 in A1
54          MOVE         A1,A2                ;
55          NEG         A2                    ;XY displacement -r::0 in A2
56          MOVE         A2,A3                ;
57          SRL         16,A3                 ;XY displacement 0::-r in A3
58          ADDXY         A0,A1               ;A1 = XY pointer in octant 2
59          ADDXY         A0,A2               ;A2 = XY pointer in octant 5
60          ADDXY         A0,A3               ;
```

```
61          ADDXY       A8,A0           ;A0 = XY pointer in octant 0
62          MOVE        A1,B0           ;B0 = XY pointer in octant 1
63          MOVE        A3,B1           ;B1 = XY pointer in octant 3
64          MOVE        A3,B2           ;B2 = XY pointer in octant 4
65          MOVE        A2,B7           ;B7 = XY pointer in octant 6
66          MOVE        A0,A3           ;A3 = XY pointer in octant 7
67  * Set up XY increments for horizontal, vertical and diagonal moves.
68          MOVI        >00010000,A10       ;deltay = +1, deltax =  0
69          MOVI        >0001FFFF,A11       ;deltay = +1, deltax = -1
70          MOVI        >0000FFFF,A12       ;deltay =  0, deltax = -1
71          MOVI        -1,A13          ;deltay = -1, deltax = -1
72          MOVI        >FFFF0000,A14       ;deltay = -1, deltax =  0
73          MOVE        A10,B10         ;deltay = +1, deltax =  0
74          MOVI        >00010001,B11       ;deltay = +1, deltax = +1
75          MOVK        >00000001,B12       ;deltay =  0, deltax = +1
76          MOVI        >FFFF0001,B13       ;deltay = -1, deltax = +1
77          MOVE        A14,B14         ;deltay = -1, deltax =  0
78  * Set up initial values for loop control variables.
79          MOVE        A8,A4           ;initially d = 5 - 4*r
80          SLL         2,A4            ;
81          NEG         A4              ;
82          ADDK        5,A4            ;
83          MOVK        4,A5            ;initially p = 4
84          MOVE        A8,A6           ;initially q = 4 - 8*r
85          SLL         3,A6            ;
86          NEG         A6              ;
87          ADDK        4,A6            ;
88  * Draw 4 starting pixels at top, bottom and sides of circle.
89          DRAV        A10,A0          ;draw & advance in octant 0
90          ADDXY       B12,B0          ;------ advance in octant 1
91          DRAV        A12,A1          ;draw & advance in octant 2
92          DRAV        B10,B1          ;draw & advance in octant 3
93          ADDXY       B14,B2          ;------ advance in octant 4
94          DRAV        A12,A2          ;draw & advance in octant 5
95          ADDXY       B12,B7          ;------ advance in octant 6
96          ADDXY       A14,A3          ;------ advance in octant 7
97          SUBK        1,A8            ;decrement loop count by 1
98          JRLE    C5                  ;jump if r < 2  (all done)
99          ADDK        8,A5            ;p += 8
100         ADDK        8,A6            ;q += 8
101         ADD         A5,A4           ;d += p
102         JRNN        C2              ;jump if d >= 0 (and r < 5)
103 *----------BEGIN INNER LOOP OF CIRCLE ALGORITHM-------------
104 * Move arc in horizontal or vertical direction in each of the 8 octants.
105 C1:
106         DRAV        A10,A0          ;draw & advance in octant 0
107         DRAV        B12,B0          ;draw & advance in octant 1
108         DRAV        A12,A1          ;draw & advance in octant 2
109         DRAV        B10,B1          ;draw & advance in octant 3
110         DRAV        B14,B2          ;draw & advance in octant 4
111         DRAV        A12,A2          ;draw & advance in octant 5
112         DRAV        B12,B7          ;draw & advance in octant 6
113         DRAV        A14,A3          ;draw & advance in octant 7
114         SUBK        1,A8            ;decrement loop count by 1
115         JRLE    C5                  ;jump if at end of octant
116         ADDK        8,A5            ;p += 8
117         ADDK        8,A6            ;q += 8
118         ADD         A5,A4           ;d += p
119         JRN         C1              ;jump if d < 0
120 * Move arc in diagonal direction in each of the 8 octants.
```

```
121 C2:
122         DRAV        A11,A0              ;draw & advance in octant 0
123         DRAV        B13,B0              ;draw & advance in octant 1
124         DRAV        A13,A1              ;draw & advance in octant 2
125         DRAV        B11,B1              ;draw & advance in octant 3
126         DRAV        B13,B2              ;draw & advance in octant 4
127         DRAV        A11,A2              ;draw & advance in octant 5
128         DRAV        B11,B7              ;draw & advance in octant 6
129         DRAV        A13,A3              ;draw & advance in octant 7
130         SUBK        2,A8                  ;decrement loop count by 2
131         JRLE    C4                       ;jump if at end of octant
132         ADDK        8,A5                  ;p += 8
133         ADDK        16,A6               ;q += 16
134         ADD        A6,A4                ;d += q
135         JRN        C1                   ;jump if d < 0
136         JRUC        C2                   ;jump if d >= 0
137 * In special case in which r = 0, draw a single point.
138 C3:
139         DRAV    A0,A0                    ;
140         JRUC    C5                       ;
141 * Check whether 4 final pixels must be drawn so that arcs touch.
142 C4:
143         JRNZ    C5                       ;jump if arcs already touch
144         DRAV        A8,A0               ;draw pixel between oct's 0,1
145         DRAV        A8,A1               ;draw pixel between oct's 2,3
146         DRAV        A8,A2               ;draw pixel between oct's 4,5
147         DRAV        A8,A3               ;draw pixel between oct's 6,7
148 * Restore registers and return.
149 C5:
150         MMFM        SP,B0,B1,B2,B7,B10,B11,B12,B13,B14
151         MMFM        SP,A0,A1,A2,A3,A4,A5,A6,A10,A11,A12,A13,A14
152         SUBI        3*32,A14              ;adjust parameter stack ptr
153         RETS        2                   ;return
154         .end
```

PROGRAMME DE:

TRACE D'UN QUART DE CERCLE

```
1  #include "colors.h"
2
3  main()
4  {
5     int r,x,y,i;
6     long color;
7
8     init_sdb();
9     init_palet();
10    erase_screen();
11    color = 0x77777777;
12    x=350;
13    y=250;
14    r=200;
15        set_color1(color);
16     for (i=1;i<=100;i++){
17      circles(r,x,y);
18     }
19    asm ("        TRAP 29");
20    printf ("ok !!!!\n");
21    for (;;);
22    }
```

```
1          .width    132
2
3  *         (c) Copyright 1987, Texas Instruments, Incorporated
4  *---------------------------------------------------------------------
5  *---------------  TMS34010 Graphics Function Library  ----------------
6  *---------------------------------------------------------------------
7  * circle function
8  *
9  *   Draw outline of circle.  Radius of circle is specified by argument
10 *   r.        Center point is specified by arguments x and y.
11 *
12 *   This function is designed to be called from a GSP-C program.
13 *---------------------------------------------------------------------
14 * Usage:  circle(r, x, y);
15 *
16 * Description of stack arguments:
17 *   int r;  /* radius */
18 *   int x, y;  /* center coordinates */
19 *
20 * Returned in register A8:  Void (undefined)
21 *
22 * Registers altered:  A8
23 *---------------------------------------------------------------------
24 * Revision history:
25 *   1/9/87.....Original version written.................Jerry Van Aken
26 *---------------------------------------------------------------------
27 ;
28         .title   'circles'
29         .file         'circles.asm'
30 ;
31 ;
32 ;     DECLARE GLOBAL FUNCTION NAME
33 ;
34         .globl    _circles
35 ;
36 ;
37 ;     ENTRY POINT
38 ;
39 _circles:
40
41         MMTM          SP,A0,A1,A4,A5,A6,A10,A11,A12,A13,A14
42         MMTM          SP,B0,B1,B2,B10,B11,B12,B13
43         MOVE          *-A14,A8,1              ;get argument 'r'
44         MOVE          *-A14,A0,1              ;get argument 'x'
45         MOVE          *-A14,A1,1              ;get argument 'y'
46         TRAP     29
47 * Form XY address of circle's center point.
48         SLL          16,A1                ;concatenate x and y
49         MOVY          A1,A0                ;
50 * Get starting XY addresses for arcs in 8 octants.
51         MOVE          A8,A1                ;copy r
52         JRZ      C3                        ;jump if r = 0
53         SLL          16,A1                ;XY displacement +r::0 in A1
54         MOVE          A1,B0                ;
55         NEG          B0                    ;XY displacement -r::0 in A2
56         MOVE          B0,B1                ;
57         SRL          16,B1                ;XY displacement 0::-r in A3
58         MOVE     A0,B2
59         ADDXY          A0,A1               ;A1 = XY pointer in octant 2
60         ADDXY          B2,B0               ;A2 = XY pointer in octant 5
```

```
61          ADDXY       B2,B1              ;
62          ADDXY       A8,A0              ;A0 = XY pointer in octant 0
63  * Set up XY increments for horizontal, vertical and diagonal moves.
64          MOVI        >00010000,A10        ;deltay = +1, deltax =  0
65          MOVI        >0001FFFF,A11        ;deltay = +1, deltax = -1
66          MOVI        >0000FFFF,A12        ;deltay =  0, deltax = -1
67          MOVI        -1,A13              ;deltay = -1, deltax = -1
68          MOVI        >FFFF0000,B10        ;deltay = -1, deltax =  0
69          MOVI        >00010001,B11        ;deltay = +1, deltax = +1
70          MOVK        >00000001,B12        ;deltay =  0, deltax = +1
71          MOVI        >FFFF0001,B13        ;deltay = -1, deltax = +1
72  * Set up initial values for loop control variables.
73          SLL     1,A8
74          MOVE        A8,A4              ;initially d = 5 - 4*r
75          SLL       2,A4                 ;
76          NEG       A4                 ;
77          ADDK       5,A4                ;
78          MOVK       4,A5                ;initially p = 4
79          MOVE       A8,A6               ;initially q = 4 - 8*r
80          SLL       3,A6                 ;
81          NEG       A6                 ;
82          ADDK       4,A6                ;
83  * Draw 4 starting pixels at top, bottom and sides of circle.
84          DRAV       A10,A0              ;draw & advance in octant 0
85          DRAV       A12,A1              ;draw & advance in octant 2
86          DRAV       B10,B1              ;draw & advance in octant 3
87          DRAV       B12,B0              ;draw & advance in octant 5
88          SUBK       1,A8                  ;decrement loop count by 1
89          JRLE    C5                     ;jump if r < 2  (all done)
90          ADDK       8,A5                  ;p += 8
91          ADDK       8,A6                  ;q += 8
92          ADD       A5,A4                ;d += p
93          JRNN       C2                  ;jump if d >= 0 (and r < 5)
94  *----------BEGIN INNER LOOP OF CIRCLE ALGORITHM-------------
95  * Move arc in horizontal or vertical direction in each of the 8 octants.
96  C1:
97          DRAV       A10,A0              ;draw & advance in octant 0
98          DRAV       B12,B0              ;draw & advance in octant 1
99          DRAV       A12,A1              ;draw & advance in octant 2
100         DRAV       B10,B1              ;draw & advance in octant 3
101         SUBK       1,A8                  ;decrement loop count by 1
102         JRLE    C5                     ;jump if at end of octant
103         ADDK       8,A5                  ;p += 8
104         ADDK       8,A6                  ;q += 8
105         ADD       A5,A4                ;d += p
106         JRN       C1                   ;jump if d < 0
107 * Move arc in diagonal direction in each of the 8 octants.
108 C2:
109         DRAV       A11,A0              ;draw & advance in octant 0
110         DRAV       B11,B0              ;draw & advance in octant 1
111         DRAV       A13,A1              ;draw & advance in octant 2
112         DRAV       B13,B1              ;draw & advance in octant 3
113         SUBK       2,A8                  ;decrement loop count by 2
114         JRLE    C4                     ;jump if at end of octant
115         ADDK       8,A5                  ;p += 8
116         ADDK       16,A6               ;q += 16
117         ADD       A6,A4                ;d += q
118         JRN       C1                   ;jump if d < 0
119         JRUC       C2                   ;jump if d >= 0
120 * In special case in which r = 0, draw a single point.
```

```
121 C3:
122         DRAV    A0,A0                   ;
123         JRUC    C5                      ;
124 * Check whether 4 final pixels must be drawn so that arcs touch.
125 C4:
126         JRNZ    C5                      ;jump if arcs already touch
127         DRAV        A8,A0               ;draw pixel between oct's 0,1
128         DRAV        A8,A1               ;draw pixel between oct's 2,3
129         DRAV        A8,A2               ;draw pixel between oct's 4,5
130         DRAV        A8,A3               ;draw pixel between oct's 6,7
131 * Restore registers and return.
132 C5:
133         MMFM        SP,B0,B1,B2,B10,B11,B12,B13
134         MMFM        SP,A0,A1,A4,A5,A6,A10,A11,A12,A13,A14
135         SUBI        3*32,A14              ;adjust parameter stack ptr
136         RETS        2                   ;return
137         .end
```

PROGRAMME DE:

REMPLISSAGE

```
#include "colors.h"
   int max(tab,n)
   int tab[],n;
   { int m,i;
   m=0;
   for (i=1;i<=n;i++) {
   if (tab[i]>m)  m=tab[i];  }
   return m;
   }
   tl(yi,yf,dyi)
   int yi,yf,*dyi;
   {
   if ((yf-yi)>0) *dyi=1;
   else if ((yf-yi)<0) *dyi= -1;
   else *dyi=0;
   }
   segment1(xx0,yy0,xx1,yy1,m)
   int xx0,yy0,xx1,yy1,m;
   { long color;
   remp(xx0,yy0,xx1,yy1,m,1);
   remp(xx0,yy0,xx1,yy1,m,2);
   remp(xx0,yy0,xx1,yy1,m,3);
   remp(xx0,yy0,xx1,yy1,m,4);
   remps(xx0+1,yy0,xx1+1,yy1,m,1);
   remps(xx0+1,yy0,xx1+1,yy1,m,2);
   remps(xx0+1,yy0,xx1+1,yy1,m,3);
   remps(xx0+1,yy0,xx1+1,yy1,m,4);
   }
   remplis(x0,y0,x1,y1,x2,y2,x3,y3,m)
   int x0,y0,x1,y1,x2,y2,x3,y3,m;
   { int dy0,dy1,dx1,dy2;
   tl(y0,y1,&dy0);
   tl(y1,y2,&dy1);
   tl(x1,x2,&dx1);
   tl(y2,y3,&dy2);
   if (dy1==0)
   {
    if ((dy0==1)&&(dy2==1))
    {
       if (dx1==1)
     {
   segment1(x2,y2,x2,y2,m); }
   else {
   segment1(x1,y1,x1,y1,m); } }
     if ((dy0== -1)&&(dy2== -1))
   {
       if (dx1==1)
     {
   segment1(x1+1,y1,x1+1,y1,m); }
   else {
   segment1(x2+1,y2,x2+1,y2,m); } } }
   else {
   if (dy0==1)
   {
     if (dy1==1)
     {
       segment1(x1,y1,x1,y1,m);
       segment1(x1,y1,x2,y2,m); }
   else {
          segment1(x1,y1,x2,y2,m); } }
```

```
61     else {
62           if (dy0== -1)
63     {
64       if (dy1== -1)
65       {
66     segment1(x1+1,y1,x1+1,y1,m);
67         segment1(x1,y1,x2,y2,m); }
68      else {
69            segment1(x1,y1,x2,y2,m); } }
70     else {
71          if (dy1==1)
72     {
73       segment1(x1,y1,x2,y2,m); }
74         else {
75               segment1(x1,y1,x2,y2,m); } } } }
76     }
77 main()
78 {
79     int xtab[100],ytab[100];
80     int n,i,m,j;
81
82     init_sdb();
83     init_palet();
84     erase_screen();
85     n=17;
86     xtab[1]=250;
87     ytab[1]=50;
88     xtab[2]=200;
89     ytab[2]=150;
90     xtab[3]=175;
91     ytab[3]=100;
92     xtab[4]=100;
93     ytab[4]=175;
94     xtab[5]=100;
95     ytab[5]=275;
96     xtab[6]=50;
97     ytab[6]=325;
98     xtab[7]=75;
99     ytab[7]=375;
100    xtab[8]=125;
101    ytab[8]=300;
102    xtab[9]=150;
103    ytab[9]=450;
104    xtab[10]=225;
105    ytab[10]=400;
106    xtab[11]=275;
107    ytab[11]=425;
108    xtab[12]=325;
109    ytab[12]=350;
110    xtab[13]=375;
111    ytab[13]=250;
112    xtab[14]=425;
113    ytab[14]=225;
114    xtab[15]=400;
115    ytab[15]=200;
116    xtab[16]=350;
117    ytab[16]=75;
118    xtab[17]=300;
119    ytab[17]=125;
120    for (j=1;j<100;j++){
```

```
121     m=max(xtab,n);
122     remplis(xtab[n],ytab[n],xtab[1],ytab[1],xtab[2],ytab[2],
        xtab[3],ytab[3],m);
123     for (i=2;i<=n-2;i++) {
124     remplis(xtab[i-1],ytab[i-1],xtab[i],ytab[i],xtab[i+1],ytab[i+1],
        xtab[i+2],ytab[i+2],m);
125     }
126     remplis(xtab[n-2],ytab[n-2],xtab[n-1],ytab[n-1],xtab[n],ytab[n],
        xtab[1],ytab[1],m);
127     remplis(xtab[n-1],ytab[n-1],xtab[n],ytab[n],xtab[1],ytab[1],
        xtab[2],ytab[2],m);
128     }
129     asm ("        TRAP 29");
130     printf ("ok !!!!\n");
131     for (;;);
132     }
```

```
1
2  *          (c) Copyright 1986, Texas Instruments, Incorporated
3  *----------------------------------------------------------------------
4  *---------------  TMS34010 Graphics Function Library  -----------------
5  *----------------------------------------------------------------------
6  * line function
7  *
8  *   This version of the function uses the LINE instruction, and will run
9  *   only on TMX34010A samples and above (revisions 2.0 and higher).
10 *   Another version of line is provided to run on TMX34010
11 *   (revision 1.0) samples.
12 *
13 *   Draw a line from point (xs,ys) to point (xe,ye) using Bresenham's
14 *   algorithm.
15 *
16 *   This function is designed to be called from a GSPC program.
17 *----------------------------------------------------------------------
18 * Usage:  line(xs, ys, xe, ye);
19 *
20 * Description of the arguments:
21 *   int xs, ys;  /* starting line coordinates */
22 *   int xe, ye;  /* ending line coordinates   */
23 *
24 * Returned in register A8:  Void (undefined).
25 *
26 * Registers altered:  A8
27 *----------------------------------------------------------------------
28 * Revision history:
29 *   12/02/86...Original version written.................Mike Asal
30 *----------------------------------------------------------------------
31          .title   'draw a line'
32          .file    'remp.asm'
33          .nolist
34          .copy         macros.hdr
35          .list
36 ;
37 ;
38 ;      DECLARE GLOBAL FUNCTION NAME
39 ;
40          .globl    _remp
41 ;
42 ;
43 ;      ENTRY POINT
44 ;
45 _remp:
46             MMTM    SP,A0,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12,A13
47             MMTM    SP,B10,B11,B12,B13,B14
48 * Calculate XY addresses for line start and end points.
49             MOVE    *-A14,A0,1            ; Get start x
50             MOVE    *-A14,A1,1      ; Get start y
51             SLL     16,A1
52             MOVY    A1,A0            ; B2 = (y0,x0)
53             MOVE    *-A14,A1,1      ; Get end x
54             MOVE    *-A14,A2,1      ; Get end y
55             SLL     16,A2
56             MOVY    A2,A1            ; B10 = (y1,x1)
57 * Determine which octant line is in, and set up accordingly.
58             MOVE    *-A14,A2,1
59             MOVE    A2,B10
60             MOVI    >00000002,B12
```

```
61                MOVE    A1,A13
62                CLR     A2
63                MOVI    >0028,A2
64                MOVB    A2,@>C00000B8
65                CLR     A2
66                SUBXY   A0,A1             ; B10 = (y1-y0,x1-x0) = (b,a)
67                JRNZ     a1
68                JRNN    a1
69                MOVE    *-A14,A2,1
70                SUBK    1,A2
71                JRZ     fin
72                JRUC    label10
73  a1:           JRNC    bpos
74                JRNV   bneg_apos
75  bneg_aneg:    SUBXY   A1,A2              ; B7 = (|b|,|a|)
76                MOVI    >FFFEFFFE,A3             ; B11 = (-1,-1)
77                JRUC    cmp_b_a
78  bneg_apos:    SUBXY   A1,A2
79                MOVX    A1,A2             ; B7 = (|b|,|a|)
80                MOVI    >FFFE0002,A3      ; B11 = (-1,1)
81                JRUC    cmp_b_a
82  bpos:              JRXNN   bpos_apos
83  bpos_aneg:    SUBXY   A1,A2
84                MOVY    A1,A2             ; B7 = (|b|,|a|)
85                MOVI    >0002FFFE,A3     ; B11 = (1,-1)
86                JRUC    cmp_b_a
87  bpos_apos:    MOVE    A1,A2              ; B7 = (|b|,|a|)
88                MOVI    >00020002,A3     ; B11 = (1,1)
89  cmp_b_a:      CLR     A4
90                MOVE    A2,A5
91                SRL     16,A5             ; B0 = b
92                CLR     A1
93                MOVX    A2,A1             ; B10 = a
94                CMP     A5,A1
95                JRGT    a_ge_b
96  a_lt_b:       MOVE    A5,A1
97                MOVX    A2,A5
98                RL           16,A2            ; a and b swapped
99                MOVY    A3,A4
100               SLL     1,A5
101               SUB     A1,A5             ; B0 = 2b - a
102               JRUC    label0
103 * If drawing in +Y direction, use LINE 0.  Otherwise, use LINE 1.
104 a_ge_b:       MOVX    A3,A4
105               SLL     1,A5
106               SUB     A1,A5             ; B0 = 2b - a
107 label0:       SRL     1,A1
108               ADDK    1,A1
109               CLR     A6
110               MOVY    A2,A7
111               SRL     16,A7
112               MOVE    A7,A9
113               SLL     1,A7
114               MOVX    A2,A6
115               SLL     1,A6
116               NEG     A6
117               ADD     A7,A6
118               MOVE    A6,A8
119               SRA     1,A8
120               MOVE    A5,A10
```

```
121                 SRA     1,A10
122                 BTST    0,A5
123                 JRZ     label1
124                 BTST    31,A5
125                 JRZ     label1
126                 SUBK    1,A10
127 label1:         CLR     A12
128                 MOVE    *-A14,A2,1
129                 CLR     A5
130                 MOVX    A4,A5
131                 MOVE    A5,A5
132                 JRNZ    label2
133                 SUBK    1,A2
134                 JRZ     label3
135                 SUBK    1,A2
136                 JRNZ    label4
137                 MOVE    A8,A11
138                 SUB     A9,A11
139                 ADD     A11,A10
140                 BTST    15,A3
141                 JRZ     label25
142                 MOVK    1,A12
143                 SLL     16,A12
144 label25:        MOVX    A3,A12
145                 SRA     1,A12
146                 ADDXY   A12,A0
147                 JRUC    label3
148 label4:         SUBK    1,A2
149                 JRNZ    label5
150                 ADD     A9,A10
151                 MOVY    A3,A12
152                 SRA     1,A12
153                 ADDXY   A12,A0
154                 JRUC    label3
155 label5:         ADD     A8,A10
156                 MOVE    A3,A12
157                 BTST    15,A3
158                 JRZ     label26
159                 ORI     >00010000,A12
160 label26:        SRA     1,A12
161                 ADDXY   A12,A0
162                 JRUC    label3
163 label2:         SUBK    1,A2
164                 JRZ     label3
165                 SUBK    1,A2
166                 JRNZ    label6
167                 ADD     A9,A10
168                 BTST    15,A3
169                 JRZ     label27
170                 MOVK    1,A12
171                 SLL     16,A12
172 label27:        MOVX    A3,A12
173                 SRA     1,A12
174                 ADDXY   A12,A0
175                 JRUC    label3
176 label6:         SUBK    1,A2
177                 JRNZ    label5
178                 MOVE    A8,A11
179                 SUB     A9,A11
180                 ADD     A11,A10
```

```
181                MOVY    A3,A12
182                SRA     1,A12
183                ADDXY   A12,A0
184 label3:        CLR     A5
185                CMPI    0,A1
186                JRLE    label10
187 label18:       CLR     A11
188                CLR     A12
189                MOVX    A4,A11
190                MOVE    A11,A11
191                JRZ     label7
192                BTST    15,A11
193                JRNZ    label8
194                CLR     A11
195                MOVX    A0,A11
196                MOVX    A13,A12
197                CMP     A12,A11
198                JRLE    label9
199                JRUC    label10
200                CLR     A11
201 label8:        MOVX    A0,A11
202                MOVX    A13,A12
203                CMP     A11,A12
204                JRLE    label9
205                JRUC    label10
206 label7:        CLR     A11
207                MOVY    A4,A11
208                MOVE    A11,A11
209                JRN     label11
210                CLR     A11
211                MOVY    A0,A11
212                MOVY    A13,A12
213                CMP     A12,A11
214                JRLE    label9
215                JRUC    label10
216                CLR     A11
217 label11:       MOVY    A0,A11
218                MOVY    A13,A12
219                CMP     A11,A12
220                JRLE    label9
221                JRUC    label10
222 label9:        CMPI    0,A10
223                JRLT    label13
224                CMP     A8,A10
225                JRLT    label12
226                CMP     A9,A10
227                JRGE    label12
228 label30:       MOVE    A0,B11
229                MOVY    B11,B10
230 label20:       CMP     B10,B11
231                JRGT    label12
232                DRAV    B12,B11
233                JRUC    label20
234 label12:       ADDXY   A3,A0
235                CLR     A5
236 label14:       ADD     A6,A10
237                JRUC    label15
238 label13:       CMP     A8,A10
239                JRLT    label16
240                CMP     A9,A10
```

```
241                JRGE    label16
242 label31:       MOVE    A0,B11
243                MOVY    B11,B10
244 label21:       CMP     B10,B11
245                JRGT    label16
246                DRAV    B12,B11
247                JRUC    label21
248 label16:       ADDXY   A4,A0
249                MOVK    1,A5
250 label17:       ADD     A7,A10
251 label15:       SUBK    1,A1
252                JRNZ    label18
253                JRUC    label10
254 fin:           MOVE    A0,B11
255                MOVY    B11,B10
256 label22:       CMP     B10,B11
257                JRGT    label10
258                DRAV    B12,B11
259                JRUC    label22
260 label10:       MMFM    SP,B10,B11,B12,B13,B14
261                MMFM    SP,A0,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12,A13
262                RETS    2
263                .end
```

```
1
2  *          (c) Copyright 1986, Texas Instruments, Incorporated
3  *-------------------------------------------------------------------------
4  *---------------  TMS34010 Graphics Function Library  -------------------
5  *-------------------------------------------------------------------------
6  * line function
7  *
8  *   This version of the function uses the LINE instruction, and will run
9  *   only on TMX34010A samples and above (revisions 2.0 and higher).
10 *   Another version of line is provided to run on TMX34010
11 *   (revision 1.0) samples.
12 *
13 *   Draw a line from point (xs,ys) to point (xe,ye) using Bresenham's
14 *   algorithm.
15 *
16 *   This function is designed to be called from a GSPC program.
17 *-------------------------------------------------------------------------
18 * Usage:  line(xs, ys, xe, ye);
19 *
20 * Description of the arguments:
21 *   int xs, ys;  /* starting line coordinates */
22 *   int xe, ye;  /* ending line coordinates   */
23 *
24 * Returned in register A8:  Void (undefined).
25 *
26 * Registers altered:  A8
27 *-------------------------------------------------------------------------
28 * Revision history:
29 *   12/02/86...Original version written.................Mike Asal
30 *-------------------------------------------------------------------------
31          .title   'draw a line'
32          .file    'remps.asm'
33          .nolist
34          .copy         macros.hdr
35          .list
36 ;
37 ;
38 ;     DECLARE GLOBAL FUNCTION NAME
39 ;
40          .globl    _remps
41 ;
42 ;
43 ;     ENTRY POINT
44 ;
45 _remps:
46             MMTM    SP,A0,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12,A13
47             MMTM    SP,B10,B11,B12,B13,B14
48 * Calculate XY addresses for line start and end points.
49             MOVE    *-A14,A0,1            ; Get start x
50             MOVE    *-A14,A1,1      ; Get start y
51             SLL     16,A1
52             MOVY    A1,A0            ; B2 = (y0,x0)
53             MOVE    *-A14,A1,1      ; Get end x
54             MOVE    *-A14,A2,1      ; Get end y
55             SLL     16,A2
56             MOVY    A2,A1            ; B10 = (y1,x1)
57 * Determine which octant line is in, and set up accordingly.
58             MOVE    *-A14,A2,1
59             MOVE    A2,B10
60             MOVI    >00000002,B12
```

```
61                MOVE    A1,A13
62                CLR     A2
63                MOVI    >0028,A2
64                MOVB    A2,@>C00000B8
65                CLR     A2
66                SUBXY   A0,A1           ; B10 = (y1-y0,x1-x0) = (b,a)
67                JRNZ     a1
68                JRNN    a1
69                MOVE    *-A14,A2,1
70                SUBK    1,A2
71                JRZ     fin
72                JRUC    label10
73   a1:          JRNC    bpos
74                JRNV   bneg_apos
75   bneg_aneg:   SUBXY   A1,A2            ; B7 = (|b|,|a|)
76                MOVI    >FFFEFFFE,A3            ; B11 = (-1,-1)
77                JRUC    cmp_b_a
78   bneg_apos:   SUBXY   A1,A2
79                MOVX    A1,A2           ; B7 = (|b|,|a|)
80                MOVI    >FFFE0002,A3    ; B11 = (-1,1)
81                JRUC    cmp_b_a
82   bpos:             JRXNN   bpos_apos
83   bpos_aneg:   SUBXY   A1,A2
84                MOVY    A1,A2           ; B7 = (|b|,|a|)
85                MOVI    >0002FFFE,A3    ; B11 = (1,-1)
86                JRUC    cmp_b_a
87   bpos_apos:   MOVE    A1,A2            ; B7 = (|b|,|a|)
88                MOVI    >00020002,A3    ; B11 = (1,1)
89   cmp_b_a:     CLR     A4
90                MOVE    A2,A5
91                SRL     16,A5           ; B0 = b
92                CLR     A1
93                MOVX    A2,A1           ; B10 = a
94                CMP     A5,A1
95                JRGT    a_ge_b
96   a_lt_b:      MOVE    A5,A1
97                MOVX    A2,A5
98                RL           16,A2          ; a and b swapped
99                MOVY    A3,A4
100               SLL     1,A5
101               SUB     A1,A5           ; B0 = 2b - a
102               JRUC    label0
103  * If drawing in +Y direction, use LINE 0.  Otherwise, use LINE 1.
104  a_ge_b:      MOVX    A3,A4
105               SLL     1,A5
106               SUB     A1,A5           ; B0 = 2b - a
107  label0:      SRL     1,A1
108               ADDK    1,A1
109               CLR     A6
110               MOVY    A2,A7
111               SRL     16,A7
112               MOVE    A7,A9
113               SLL     1,A7
114               MOVX    A2,A6
115               SLL     1,A6
116               NEG     A6
117               ADD     A7,A6
118               MOVE    A6,A8
119               SRA     1,A8
120               MOVE    A5,A10
```

```
121                SRA     1,A10
122                BTST    0,A5
123                JRZ     label1
124                BTST    31,A5
125                JRZ     label1
126                SUBK    1,A10
127 label1:        CLR     A12
128                MOVE    *-A14,A2,1
129                CLR     A5
130                MOVX    A4,A5
131                MOVE    A5,A5
132                JRNZ    label2
133                SUBK    1,A2
134                JRZ     label3
135                SUBK    1,A2
136                JRNZ    label4
137                MOVE    A8,A11
138                SUB     A9,A11
139                ADD     A11,A10
140                BTST    15,A3
141                JRZ     label25
142                MOVK    1,A12
143                SLL     16,A12
144 label25:       MOVX    A3,A12
145                SRA     1,A12
146                ADDXY   A12,A0
147                JRUC    label3
148 label4:        SUBK    1,A2
149                JRNZ    label5
150                ADD     A9,A10
151                MOVY    A3,A12
152                SRA     1,A12
153                ADDXY   A12,A0
154                JRUC    label3
155 label5:        ADD     A8,A10
156                MOVE    A3,A12
157                BTST    15,A3
158                JRZ     label26
159                ORI     >00010000,A12
160 label26:       SRA     1,A12
161                ADDXY   A12,A0
162                JRUC    label3
163 label2:        SUBK    1,A2
164                JRZ     label3
165                SUBK    1,A2
166                JRNZ    label6
167                ADD     A9,A10
168                BTST    15,A3
169                JRZ     label27
170                MOVK    1,A12
171                SLL     16,A12
172 label27:       MOVX    A3,A12
173                SRA     1,A12
174                ADDXY   A12,A0
175                JRUC    label3
176 label6:        SUBK    1,A2
177                JRNZ    label5
178                MOVE    A8,A11
179                SUB     A9,A11
180                ADD     A11,A10
```

```
181                MOVY    A3,A12
182                SRA     1,A12
183                ADDXY   A12,A0
184 label3:        CLR     A5
185                CMPI    0,A1
186                JRLE    label10
187 label18:       CLR     A11
188                CLR     A12
189                MOVX    A4,A11
190                MOVE    A11,A11
191                JRZ     label7
192                BTST    15,A11
193                JRNZ    label8
194                CLR     A11
195                MOVX    A0,A11
196                MOVX    A13,A12
197                CMP     A12,A11
198                JRLE    label9
199                JRUC    label10
200                CLR     A11
201 label8:        MOVX    A0,A11
202                MOVX    A13,A12
203                CMP     A11,A12
204                JRLE    label9
205                JRUC    label10
206 label7:        CLR     A11
207                MOVY    A4,A11
208                MOVE    A11,A11
209                JRN     label11
210                CLR     A11
211                MOVY    A0,A11
212                MOVY    A13,A12
213                CMP     A12,A11
214                JRLE    label9
215                JRUC    label10
216                CLR     A11
217 label11:       MOVY    A0,A11
218                MOVY    A13,A12
219                CMP     A11,A12
220                JRLE    label9
221                JRUC    label10
222 label9:        CMPI    0,A10
223                JRLT    label13
224                CMP     A8,A10
225                JRLT    label12
226                CMP     A9,A10
227                JRGE    label12
228 label30:       CVXYL   A0,A2
229                MOVE    A2,B14
230                MOVB    *B14,B14
231                MOVE    A0,A2
232                SUBK    1,A2
233                CVXYL   A2,A2
234                MOVE    A2,B13
235                MOVB    *B13,B13
236                CMP     B13,B14
237                JRZ     label12
238                MOVE    A0,B11
239                MOVY    B11,B10
240 label20:       CMP     B10,B11
```

```
241                JRGT    label12
242                DRAV    B12,B11
243                JRUC    label20
244 label12:       ADDXY   A3,A0
245                CLR     A5
246 label14:       ADD     A6,A10
247                JRUC    label15
248 label13:       CMP     A8,A10
249                JRLT    label16
250                CMP     A9,A10
251                JRGE    label16
252 label31:       CVXYL   A0,A2
253                MOVE    A2,B14
254                MOVB    *B14,B14
255                MOVE    A0,A2
256                SUBK    1,A2
257                CVXYL   A2,A2
258                MOVE    A2,B13
259                MOVB    *B13,B13
260                CMP     B13,B14
261                JRZ     label16
262                MOVE    A0,B11
263                MOVY    B11,B10
264 label21:       CMP     B10,B11
265                JRGT    label16
266                DRAV    B12,B11
267                JRUC    label21
268 label16:       ADDXY   A4,A0
269                MOVK    1,A5
270 label17:       ADD     A7,A10
271 label15:       SUBK    1,A1
272                JRNZ    label18
273                JRUC    label10
274 fin:           MOVE    A0,B11
275                MOVY    B11,B10
276 label22:       CMP     B10,B11
277                JRGT    label10
278                DRAV    B12,B11
279                JRUC    label22
280 label10:       MMFM    SP,B10,B11,B12,B13,B14
281                MMFM    SP,A0,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12,A13
282                RETS    2
283                .end
```

PROGRAMME DE:

REMPLISSAGE PARALLELE

```
#include "colors.h"
   int max(tab,n)
   int tab[],n;
   { int m,i;
   m=0;
   for (i=1;i<=n;i++) {
   if (tab[i]>m)  m=tab[i];  }
   return m;
   }
   t1(yi,yf,dyi)
   int yi,yf,*dyi;
   {
   if ((yf-yi)>0) *dyi=1;
   else if ((yf-yi)<0) *dyi= -1;
   else *dyi=0;
   }
   segment1(xx0,yy0,xx1,yy1,m)
   int xx0,yy0,xx1,yy1,m;
   { long color;
   remp(xx0,yy0,xx1,yy1,m,1);
   remps(xx0+1,yy0,xx1+1,yy1,m,1);
   }
   remplis(x0,y0,x1,y1,x2,y2,x3,y3,m)
   int x0,y0,x1,y1,x2,y2,x3,y3,m;
   { int dy0,dy1,dx1,dy2;
   t1(y0,y1,&dy0);
   t1(y1,y2,&dy1);
   t1(x1,x2,&dx1);
   t1(y2,y3,&dy2);
   if (dy1==0)
   {
    if ((dy0==1)&&(dy2==1))
    {
       if (dx1==1)
     {
   segment1(x2,y2,x2,y2,m); }
   else {
   segment1(x1,y1,x1,y1,m); } }
     if ((dy0== -1)&&(dy2== -1))
   {
       if (dx1==1)
     {
   segment1(x1+1,y1,x1+1,y1,m); }
   else {
   segment1(x2+1,y2,x2+1,y2,m); } } }
   else {
   if (dy0==1)
   {
     if (dy1==1)
     {
       segment1(x1,y1,x1,y1,m);
       segment1(x1,y1,x2,y2,m); }
   else {
          segment1(x1,y1,x2,y2,m); } }
   else {
         if (dy0== -1)
   {
     if (dy1== -1)
     {
   segment1(x1+1,y1,x1+1,y1,m);
```

```
61          segment1(x1,y1,x2,y2,m); }
62       else {
63             segment1(x1,y1,x2,y2,m); } }
64      else {
65           if (dy1==1)
66      {
67        segment1(x1,y1,x2,y2,m); }
68          else {
69                segment1(x1,y1,x2,y2,m); } } } }
70      }
71  main()
72  {
73      int xtab[100],ytab[100];
74      int n,i,m,j;
75
76      init_sdb();
77      init_palet();
78      erase_screen();
79      n=17;
80      xtab[1]=250;
81      ytab[1]=50;
82      xtab[2]=200;
83      ytab[2]=150;
84      xtab[3]=175;
85      ytab[3]=100;
86      xtab[4]=100;
87      ytab[4]=175;
88      xtab[5]=100;
89      ytab[5]=275;
90      xtab[6]=50;
91      ytab[6]=325;
92      xtab[7]=75;
93      ytab[7]=375;
94      xtab[8]=125;
95      ytab[8]=300;
96      xtab[9]=150;
97      ytab[9]=450;
98      xtab[10]=225;
99      ytab[10]=400;
100     xtab[11]=275;
101     ytab[11]=425;
102     xtab[12]=325;
103     ytab[12]=350;
104     xtab[13]=375;
105     ytab[13]=250;
106     xtab[14]=425;
107     ytab[14]=225;
108     xtab[15]=400;
109     ytab[15]=200;
110     xtab[16]=350;
111     ytab[16]=75;
112     xtab[17]=300;
113     ytab[17]=125;
114     for (j=1;j<=100;j++){
115     m=max(xtab,n);
116     remplis(xtab[n],ytab[n],xtab[1],ytab[1],xtab[2],ytab[2],
        xtab[3],ytab[3],m);
117     for (i=2;i<=n-2;i++) {
118     remplis(xtab[i-1],ytab[i-1],xtab[i],ytab[i],xtab[i+1],ytab[i+1],
        xtab[i+2],ytab[i+2],m);
```

```
119     }
120     remplis(xtab[n-2],ytab[n-2],xtab[n-1],ytab[n-1],xtab[n],ytab[n],
        xtab[1],ytab[1],m);
121     remplis(xtab[n-1],ytab[n-1],xtab[n],ytab[n],xtab[1],ytab[1],
        xtab[2],ytab[2],m);
122     }
123     asm ("        TRAP 29");
124     printf ("ok !!!!\n");
125     for (;;);
126     }
```

```
1
2  *          (c) Copyright 1986, Texas Instruments, Incorporated
3  *----------------------------------------------------------------------
4  *--------------- TMS34010 Graphics Function Library  -----------------
5  *----------------------------------------------------------------------
6  * line function
7  *
8  *   This version of the function uses the LINE instruction, and will run
9  *   only on TMX34010A samples and above (revisions 2.0 and higher).
10 *   Another version of line is provided to run on TMX34010
11 *   (revision 1.0) samples.
12 *
13 *   Draw a line from point (xs,ys) to point (xe,ye) using Bresenham's
14 *   algorithm.
15 *
16 *   This function is designed to be called from a GSPC program.
17 *----------------------------------------------------------------------
18 * Usage:  line(xs, ys, xe, ye);
19 *
20 * Description of the arguments:
21 *   int xs, ys;  /* starting line coordinates */
22 *   int xe, ye;  /* ending line coordinates   */
23 *
24 * Returned in register A8:  Void (undefined).
25 *
26 * Registers altered:  A8
27 *----------------------------------------------------------------------
28 * Revision history:
29 *   12/02/86...Original version written.................Mike Asal
30 *----------------------------------------------------------------------
31          .title   'draw a line'
32          .file    'remp.asm'
33          .nolist
34          .copy         macros.hdr
35          .list
36 ;
37 ;
38 ;     DECLARE GLOBAL FUNCTION NAME
39 ;
40          .globl    _remp
41 ;
42 ;
43 ;     ENTRY POINT
44 ;
45 _remp:
46             MMTM    SP,A0,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12,A13
47             MMTM    SP,B10,B11,B12,B13,B14
48 * Calculate XY addresses for line start and end points.
49             MOVE    *-A14,A0,1            ; Get start x
50             MOVE    *-A14,A1,1      ; Get start y
51             SLL     16,A1
52             MOVY    A1,A0            ; B2 = (y0,x0)
53             MOVE    *-A14,A1,1      ; Get end x
54             MOVE    *-A14,A2,1      ; Get end y
55             SLL     16,A2
56             MOVY    A2,A1            ; B10 = (y1,x1)
57 * Determine which octant line is in, and set up accordingly.
58             MOVE    *-A14,A2,1
59             MOVE    A2,B10
60             MOVI    >00000002,B12
```

```
61                MOVE    A1,A13
62                CLR     A2
63                MOVI    >0028,A2
64                MOVB    A2,@>C00000B8
65                CLR     A2
66                SUBXY   A0,A1             ; B10 = (y1-y0,x1-x0) = (b,a)
67                JRNZ     a1
68                JRNN    a1
69                MOVE    *-A14,A2,1
70                SUBK    1,A2
71                JRZ     fin
72                JRUC    label10
73   a1:          JRNC    bpos
74                JRNV   bneg_apos
75   bneg_aneg:   SUBXY   A1,A2              ; B7 = (|b|,|a|)
76                MOVI    >FFFEFFFE,A3             ; B11 = (-1,-1)
77                JRUC    cmp_b_a
78   bneg_apos:   SUBXY   A1,A2
79                MOVX    A1,A2             ; B7 = (|b|,|a|)
80                MOVI    >FFFE0002,A3      ; B11 = (-1,1)
81                JRUC    cmp_b_a
82   bpos:             JRXNN   bpos_apos
83   bpos_aneg:   SUBXY   A1,A2
84                MOVY    A1,A2             ; B7 = (|b|,|a|)
85                MOVI    >0002FFFE,A3     ; B11 = (1,-1)
86                JRUC    cmp_b_a
87   bpos_apos:   MOVE    A1,A2              ; B7 = (|b|,|a|)
88                MOVI    >00020002,A3     ; B11 = (1,1)
89   cmp_b_a:     CLR     A4
90                MOVE    A2,A5
91                SRL     16,A5             ; B0 = b
92                CLR  .  A1
93                MOVX    A2,A1             ; B10 = a
94                CMP     A5,A1
95                JRGT    a_ge_b
96   a_lt_b:      MOVE    A5,A1
97                MOVX    A2,A5
98                RL           16,A2            ; a and b swapped
99                MOVY    A3,A4
100               SLL     1,A5
101               SUB     A1,A5             ; B0 = 2b - a
102               JRUC    label0
103  * If drawing in +Y direction, use LINE 0.  Otherwise, use LINE 1.
104  a_ge_b:      MOVX    A3,A4
105               SLL     1,A5
106               SUB     A1,A5             ; B0 = 2b - a
107  label0:      SRL     1,A1
108               ADDK    1,A1
109               CLR     A6
110               MOVY    A2,A7
111               SRL     16,A7
112               MOVE    A7,A9
113               SLL     1,A7
114               MOVX    A2,A6
115               SLL     1,A6
116               NEG     A6
117               ADD     A7,A6
118               MOVE    A6,A8
119               SRA     1,A8
120               MOVE    A5,A10
```

```
121                SRA     1,A10
122                BTST    0,A5
123                JRZ     label1
124                BTST    31,A5
125                JRZ     label1
126                SUBK    1,A10
127 label1:        CLR     A12
128                MOVE    *-A14,A2,1
129                CLR     A5
130                MOVX    A4,A5
131                MOVE    A5,A5
132                JRNZ    label2
133                SUBK    1,A2
134                JRZ     label3
135                SUBK    1,A2
136                JRNZ    label4
137                MOVE    A8,A11
138                SUB     A9,A11
139                ADD     A11,A10
140                BTST    15,A3
141                JRZ     label25
142                MOVK    1,A12
143                SLL     16,A12
144 label25:       MOVX    A3,A12
145                SRA     1,A12
146                ADDXY   A12,A0
147                JRUC    label3
148 label4:        SUBK    1,A2
149                JRNZ    label5
150                ADD     A9,A10
151                MOVY    A3,A12
152                SRA     1,A12
153                ADDXY   A12,A0
154                JRUC    label3
155 label5:        ADD     A8,A10
156                MOVE    A3,A12
157                BTST    15,A3
158                JRZ     label26
159                ORI     >00010000,A12
160 label26:       SRA     1,A12
161                ADDXY   A12,A0
162                JRUC    label3
163 label2:        SUBK    1,A2
164                JRZ     label3
165                SUBK    1,A2
166                JRNZ    label6
167                ADD     A9,A10
168                BTST    15,A3
169                JRZ     label27
170                MOVK    1,A12
171                SLL     16,A12
172 label27:       MOVX    A3,A12
173                SRA     1,A12
174                ADDXY   A12,A0
175                JRUC    label3
176 label6:        SUBK    1,A2
177                JRNZ    label5
178                MOVE    A8,A11
179                SUB     A9,A11
180                ADD     A11,A10
```

```
181                MOVY    A3,A12
182                SRA     1,A12
183                ADDXY   A12,A0
184 label3:        CLR     A5
185                CMPI    0,A1
186                JRLE    label10
187 label18:       CLR     A11
188                CLR     A12
189                MOVX    A4,A11
190                MOVE    A11,A11
191                JRZ     label7
192                BTST    15,A11
193                JRNZ    label8
194                CLR     A11
195                MOVX    A0,A11
196                MOVX    A13,A12
197                CMP     A12,A11
198                JRLE    label9
199                JRUC    label10
200                CLR     A11
201 label8:        MOVX    A0,A11
202                MOVX    A13,A12
203                CMP     A11,A12
204                JRLE    label9
205                JRUC    label10
206 label7:        CLR     A11
207                MOVY    A4,A11
208                MOVE    A11,A11
209                JRN     label11
210                CLR     A11
211                MOVY    A0,A11
212                MOVY    A13,A12
213                CMP     A12,A11
214                JRLE    label9
215                JRUC    label10
216                CLR     A11
217 label11:       MOVY    A0,A11
218                MOVY    A13,A12
219                CMP     A11,A12
220                JRLE    label9
221                JRUC    label10
222 label9:        CMPI    0,A10
223                JRLT    label13
224                CMP     A8,A10
225                JRLT    label12
226                CMP     A9,A10
227                JRGE    label12
228 label30:       MOVE    A0,B11
229                MOVY    B11,B10
230 label20:       CMP     B10,B11
231                JRGT    label12
232                DRAV    B12,B11
233                JRUC    label20
234 label12:       ADDXY   A3,A0
235                CLR     A5
236 label14:       ADD     A6,A10
237                JRUC    label15
238 label13:       CMP     A8,A10
239                JRLT    label16
240                CMP     A9,A10
```

```
241                 JRGE    label16
242 label31:        MOVE    A0,B11
243                 MOVY    B11,B10
244 label21:        CMP     B10,B11
245                 JRGT    label16
246                 DRAV    B12,B11
247                 JRUC    label21
248 label16:        ADDXY   A4,A0
249                 MOVK    1,A5
250 label17:        ADD     A7,A10
251 label15:        SUBK    1,A1
252                 JRNZ    label18
253                 JRUC    label10
254 fin:            MOVE    A0,B11
255                 MOVY    B11,B10
256 label22:        CMP     B10,B11
257                 JRGT    label10
258                 DRAV    B12,B11
259                 JRUC    label22
260 label10:        MMFM    SP,B10,B11,B12,B13,B14
261                 MMFM    SP,A0,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12,A13
262                 RETS    2
263                 .end
```

```
1
2  *          (c) Copyright 1986, Texas Instruments, Incorporated
3  *----------------------------------------------------------------------
4  *---------------  TMS34010 Graphics Function Library  -----------------
5  *----------------------------------------------------------------------
6  * line function
7  *
8  *   This version of the function uses the LINE instruction, and will run
9  *   only on TMX34010A samples and above (revisions 2.0 and higher).
10 *   Another version of line is provided to run on TMX34010
11 *   (revision 1.0) samples.
12 *
13 *   Draw a line from point (xs,ys) to point (xe,ye) using Bresenham's
14 *   algorithm.
15 *
16 *   This function is designed to be called from a GSPC program.
17 *----------------------------------------------------------------------
18 * Usage:  line(xs, ys, xe, ye);
19 *
20 * Description of the arguments:
21 *   int xs, ys;  /* starting line coordinates */
22 *   int xe, ye;  /* ending line coordinates   */
23 *
24 * Returned in register A8:  Void (undefined).
25 *
26 * Registers altered:  A8
27 *----------------------------------------------------------------------
28 * Revision history:
29 *   12/02/86...Original version written.................Mike Asal
30 *----------------------------------------------------------------------
31          .title   'draw a line'
32          .file    'remps.asm'
33          .nolist
34          .copy         macros.hdr
35          .list
36 ;
37 ;
38 ;     DECLARE GLOBAL FUNCTION NAME
39 ;
40          .globl    _remps
41 ;
42 ;
43 ;     ENTRY POINT
44 ;
45 _remps:
46             MMTM    SP,A0,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12,A13
47             MMTM    SP,B10,B11,B12,B13,B14
48 * Calculate XY addresses for line start and end points.
49             MOVE    *-A14,A0,1              ; Get start x
50             MOVE    *-A14,A1,1      ; Get start y
51             SLL     16,A1
52             MOVY    A1,A0            ; B2 = (y0,x0)
53             MOVE    *-A14,A1,1      ; Get end x
54             MOVE    *-A14,A2,1      ; Get end y
55             SLL     16,A2
56             MOVY    A2,A1            ; B10 = (y1,x1)
57 * Determine which octant line is in, and set up accordingly.
58             MOVE    *-A14,A2,1
59             MOVE    A2,B10
60             MOVI    >00000002,B12
```

```
61                MOVE    A1,A13
62                CLR     A2
63                MOVI    >0028,A2
64                MOVB    A2,@>C00000B8
65                CLR     A2
66                SUBXY   A0,A1             ; B10 = (y1-y0,x1-x0) = (b,a)
67                JRNZ     a1
68                JRNN    a1
69                MOVE    *-A14,A2,1
70                SUBK    1,A2
71                JRZ     fin
72                JRUC    label10
73  a1:           JRNC    bpos
74                JRNV   bneg_apos
75  bneg_aneg:    SUBXY   A1,A2              ; B7 = (|b|,|a|)
76                MOVI    >FFFEFFFE,A3             ; B11 = (-1,-1)
77                JRUC    cmp_b_a
78  bneg_apos:    SUBXY   A1,A2
79                MOVX    A1,A2             ; B7 = (|b|,|a|)
80                MOVI    >FFFE0002,A3      ; B11 = (-1,1)
81                JRUC    cmp_b_a
82  bpos:              JRXNN   bpos_apos
83  bpos_aneg:    SUBXY   A1,A2
84                MOVY    A1,A2             ; B7 = (|b|,|a|)
85                MOVI    >0002FFFE,A3     ; B11 = (1,-1)
86                JRUC    cmp_b_a
87  bpos_apos:    MOVE    A1,A2              ; B7 = (|b|,|a|)
88                MOVI    >00020002,A3     ; B11 = (1,1)
89  cmp_b_a:      CLR     A4
90                MOVE    A2,A5
91                SRL     16,A5             ; B0 = b
92                CLR     A1
93                MOVX    A2,A1             ; B10 = a
94                CMP     A5,A1
95                JRGT    a_ge_b
96  a_lt_b:       MOVE    A5,A1
97                MOVX    A2,A5
98                RL           16,A2           ; a and b swapped
99                MOVY    A3,A4
100               SLL     1,A5
101               SUB     A1,A5             ; B0 = 2b - a
102               JRUC    label0
103 * If drawing in +Y direction, use LINE 0.  Otherwise, use LINE 1.
104 a_ge_b:       MOVX    A3,A4
105               SLL     1,A5
106               SUB     A1,A5             ; B0 = 2b - a
107 label0:       SRL     1,A1
108               ADDK    1,A1
109               CLR     A6
110               MOVY    A2,A7
111               SRL     16,A7
112               MOVE    A7,A9
113               SLL     1,A7
114               MOVX    A2,A6
115               SLL     1,A6
116               NEG     A6
117               ADD     A7,A6
118               MOVE    A6,A8
119               SRA     1,A8
120               MOVE    A5,A10
```

```
121                 SRA     1,A10
122                 BTST    0,A5
123                 JRZ     label1
124                 BTST    31,A5
125                 JRZ     label1
126                 SUBK    1,A10
127 label1:         CLR     A12
128                 MOVE    *-A14,A2,1
129                 CLR     A5
130                 MOVX    A4,A5
131                 MOVE    A5,A5
132                 JRNZ    label2
133                 SUBK    1,A2
134                 JRZ     label3
135                 SUBK    1,A2
136                 JRNZ    label4
137                 MOVE    A8,A11
138                 SUB     A9,A11
139                 ADD     A11,A10
140                 BTST    15,A3
141                 JRZ     label25
142                 MOVK    1,A12
143                 SLL     16,A12
144 label25:        MOVX    A3,A12
145                 SRA     1,A12
146                 ADDXY   A12,A0
147                 JRUC    label3
148 label4:         SUBK    1,A2
149                 JRNZ    label5
150                 ADD     A9,A10
151                 MOVY    A3,A12
152                 SRA     1,A12
153                 ADDXY   A12,A0
154                 JRUC    label3
155 label5:         ADD     A8,A10
156                 MOVE    A3,A12
157                 BTST    15,A3
158                 JRZ     label26
159                 ORI     >00010000,A12
160 label26:        SRA     1,A12
161                 ADDXY   A12,A0
162                 JRUC    label3
163 label2:         SUBK    1,A2
164                 JRZ     label3
165                 SUBK    1,A2
166                 JRNZ    label6
167                 ADD     A9,A10
168                 BTST    15,A3
169                 JRZ     label27
170                 MOVK    1,A12
171                 SLL     16,A12
172 label27:        MOVX    A3,A12
173                 SRA     1,A12
174                 ADDXY   A12,A0
175                 JRUC    label3
176 label6:         SUBK    1,A2
177                 JRNZ    label5
178                 MOVE    A8,A11
179                 SUB     A9,A11
180                 ADD     A11,A10
```

```
181                MOVY    A3,A12
182                SRA     1,A12
183                ADDXY   A12,A0
184 label3:        CLR     A5
185                CMPI    0,A1
186                JRLE    label10
187 label18:       CLR     A11
188                CLR     A12
189                MOVX    A4,A11
190                MOVE    A11,A11
191                JRZ     label7
192                BTST    15,A11
193                JRNZ    label8
194                CLR     A11
195                MOVX    A0,A11
196                MOVX    A13,A12
197                CMP     A12,A11
198                JRLE    label9
199                JRUC    label10
200                CLR     A11
201 label8:        MOVX    A0,A11
202                MOVX    A13,A12
203                CMP     A11,A12
204                JRLE    label9
205                JRUC    label10
206 label7:        CLR     A11
207                MOVY    A4,A11
208                MOVE    A11,A11
209                JRN     label11
210                CLR     A11
211                MOVY    A0,A11
212                MOVY    A13,A12
213                CMP     A12,A11
214                JRLE    label9
215                JRUC    label10
216                CLR     A11
217 label11:       MOVY    A0,A11
218                MOVY    A13,A12
219                CMP     A11,A12
220                JRLE    label9
221                JRUC    label10
222 label9:        CMPI    0,A10
223                JRLT    label13
224                CMP     A8,A10
225                JRLT    label12
226                CMP     A9,A10
227                JRGE    label12
228 label30:       CVXYL   A0,A2
229                MOVE    A2,B14
230                MOVB    *B14,B14
231                MOVE    A0,A2
232                SUBK    1,A2
233                CVXYL   A2,A2
234                MOVE    A2,B13
235                MOVB    *B13,B13
236                CMP     B13,B14
237                JRZ     label12
238                MOVE    A0,B11
239                MOVY    B11,B10
240 label20:       CMP     B10,B11
```

```
241                JRGT    label12
242                DRAV    B12,B11
243                JRUC    label20
244 label12:       ADDXY   A3,A0
245                CLR     A5
246 label14:       ADD     A6,A10
247                JRUC    label15
248 label13:       CMP     A8,A10
249                JRLT    label16
250                CMP     A9,A10
251                JRGE    label16
252 label31:       CVXYL   A0,A2
253                MOVE    A2,B14
254                MOVB    *B14,B14
255                MOVE    A0,A2
256                SUBK    1,A2
257                CVXYL   A2,A2
258                MOVE    A2,B13
259                MOVB    *B13,B13
260                CMP     B13,B14
261                JRZ     label16
262                MOVE    A0,B11
263                MOVY    B11,B10
264 label21:       CMP     B10,B11
265                JRGT    label16
266                DRAV    B12,B11
267                JRUC    label21
268 label16:       ADDXY   A4,A0
269                MOVK    1,A5
270 label17:       ADD     A7,A10
271 label15:       SUBK    1,A1
272                JRNZ    label18
273                JRUC    label10
274 fin:           MOVE    A0,B11
275                MOVY    B11,B10
276 label22:       CMP     B10,B11
277                JRGT    label10
278                DRAV    B12,B11
279                JRUC    label22
280 label10:       MMFM    SP,B10,B11,B12,B13,B14
281                MMFM    SP,A0,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12,A13
282                RETS    2
283                .end
```

BIBLIOGRAPHIE

BIBLIOGRAPHIE

* [AMD 86] - Advanced Micro Device
"Product description AM95C60 Quad Pixel Dataflow Manager (QPDM)"
Advanced Micro Device, Avril 1986

* [AMD 87] - Advanced Micro Device
"Quad Pixel Dataflow Manager (QPDM) AM95C60"
Technical Manual Revision B
Advanced Micro Device, 1987

* [ANI 87] - Intel
"82786 Hardware Configuration"
Application note - Intel - Ap 270, 1987

* [BRE 65] - J.E. Bresenham
"Algorithm for computer control of a digital plotter"
IBM System Journal, Vol. 4, N° 1, 1965

* [BRE 77] - J.E. Bresenham
"A linear algorithm for incremental digital display of circular arcs"
Graphics and image processeing, Vol. 20, N° 2, 1977

* [BRU 86] - R. BRUSQ
"Synthèse d'images par lancer de rayon (ray-tracing): la machine Cristal"
Résultats et perspectives, 2 ème colloque image
CESTA, 1986, pp. 404 - 410

* [CAR 86] - R. Carrell and J.R. Killebrew
"The TMS 34010 Graphics System Processor"
Graphics Algorithms
Byte, Décembre 1986

* [CEI 82] - CEA-EDF-INRIA
"La réalisation de logiciels graphiques interactifs"
Collection de la direction des études et recherches d'éléctricité de France
Editions Eyrolles, 1982

* [CHA 84] - B. Chazelle and J. Incerpi
"Triangulating and shape complexity"
ACM transactions on graphics, Vol. 13, N° 2, Avril 1984, pp. 135 - 152

* [CLA 80] - J. CLARK
"A vlsi geometric processor for graphics"
Computer, Juillet 1980, pp. 59 - 68

* [CLA 82] - J. CLARK
"The geometry engine: a vlsi geometry system for graphics"
Computer Graphics, Vol. 16, N° 3, Juillet 1982, pp. 127 - 133

* [CLE 83] - J.G. Cleary, B. Wyvill, G.M. Birtwistle and R. Vatt
"Multiprocessor ray tracing"
Technical Report n° 83/128/17, Université of Calgary
Alberta, Octobre 1983

* [COR 78] - V. Cordonnier
"Architecture pipeline"
Support du cours IRIA sur le traitement parallèle
Lille, Mai 1978

* [COR 81] - V. Cordonnier
"The MAP project: an associative processor for speech processing"
10 th International Conference on Parallel Processing
Bellaire, Michigan, Aout 1981

* [DIP 84] - M. Dippe and J. Swensen
"An adaptive subdivision algorithm and parallel architecture for realistic image synthesis"
Computer Graphics, Vol. 18, N° 3, 1984, pp. 149 - 158

* [DUN 83] - M.R. Dunlavey
"Efficient polygon filling algorithms for raster displays"
ACM Transactions on Graphic, Vol. 2, N° 4, Octobre 1983

* [DUR 83] - P. Durif
"Etude d'une machine parallèle de synthèse d'images à découpage par objets"
Thèse de Docteur de 3 ème cycle, Lille, Décembre 1983

* [FAK 83] - E.T. Fathi and M. Krieger
"Multiple Microprocessor Systems: What, Why, Who"
Computer, Mars 1983

* [FLY 72] - M.J. Flynn
"Some computer organisation and their effectiveness"
I.E.E.E. Transactions on Computer, Septembre 1972, pp. 59 - 76

* [FOU 84] - A. Fournier and D. Montuno
"Triangulating simple polygons and equivalent problems"
ACM Transactions on Graphic, Vol. 3, N° 2, Avril 1984

* [FUC 81] - H. Fuchs and J. Poulton
"Pixel planes: a vlsi oriented design for raster graphics engine"
Vlsi Design, N° 36, 1981, pp. 20 - 28

* [FUC 88] - H. Fuchs and AL
"Pixel planes 4: a summary"
Advances in Computer Graphics Hardware II, Springer-Verlag 1988

* [GOU 80] - H. Gouraud
"Continuous shading of curved surfaces"
Interactive Computer Graphics, 1980, pp. 302 - 308

* [HEG 85] - G. Hegron
"Synthèse d'images: algorithmes élémentaires"
Dunod Informatique, 1985, pp. 20 - 25

* [HEM 87] - F. Hemery
"Etude d'un réseau cellulaire multipipeline - simulation sur transputer en langage OCCAM"
Mémoire de D.E.A., Lille, 1987

* [HER 83] - S. Hertel and K. Melhorn
"Fast triangulating of simple polygons"
Proceedings of the 1983 International FCT Conférence,
Aout 1983, pp. 207 - 218

* [HIR 85] - S. Hiroyuki, J. Mitsuo, S. Keiji, I. Morio, J. Hiroaki, K. Masandri, H. Katsuhiko and J. Kouichi
"Fast image generation of constructive solid geometry using a cellular array processor"
Computer Graphics, Vol. 19, N° 3, 1985, pp. 95 - 102

* [INT 87] - Intel
"82786 Chmos Graphics Processor"
Advance Information - Intel - 1987

* [LAN 83] - J.M. Lane
"An algorithm for filling regions on graphics display devices"
ACM Transactions on Graphics, Vol. 2, N° 3, Juillet 1983

* [LEP 86] - E. Lepretre, M. Mériaux et A. Atamenia
"A transputer based architecture for graphics"
7 th OCCAM Users Group Workshop, 1986

* [LEP 88] - E. Lepretre et M. Mériaux
"Line drawing algorithms for cellular parallel machines"
Publication n° 63, USTLFA, Avril 1988

* [LEP 89] - E. Lepretre
"Algorithms parallèles et architectures cellulaires pour la synthèse d'images"
Thèse de Docteur en Informatique, Lille, Juin 1989

* [LOC 80] - M. Loceff
"The line"
Computer, Juin 1980, pp. 57

* [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"
Editions Eyrolles, 1982

* [MAR 82] - F. Martinez
"Vers une approche systématique de la synthèse d'images"
Thèse d'état INP, Novembre 1982

* [MAR 84] - F. Martinez
"La synthèse d'images, concepts logiciels et matériels"
Editions Editest, 1986

* [MAR 86] - F. Martinez
"Comparaison et évaluation des architectures de systèmes de synthèse d'images"
Revue de CFAO et d'infographie, Vol. 1, N° 1, 1986, pp. 45 - 62

* [MAZ 88] - G. Mazare
"Une nouvelle architecture cellulaire pour la reconstruction d'images"
PIXIM 88, Editions Hermès, Octobre 1988

* [MER 79] - M. Mériaux
"Etude et réalisation d'un terminal graphique couleur tridimensionnel fonctionnant par tâches"
Thèse de Docteur Ingénieur, Lille 1979

* [MER 84] - M. Mériaux
"Contribution à l'imagerie informatique: aspects algorithmiques et architecturaux"
Thèse de Docteur es sciences mathématiques, Lille, Juillet 1984

* [MMS 87] - Mini-Micro-Systems
"Gauging the great graphics is race"
Graphics Chip Technology
Mini-Micro-Systems, Octobre 1987, pp. 68 - 102

* [MOR 76] - P. Morvan et M. Lucas
"Images et ordinateur, introduction à l'infographie intéractive"
Editions Larrousse, Université série informatique, 1976

* [NEW 81] - W.N. Newman and R.F. Sproull
"Principles of interactive computer graphics"
Second Edition, Mac Graw Hill International, 1981

* [NIS 83] - H. Nishimura, H. Ohno, T. Kawata, I. Shirakawa and K. Omura
"LINKS-1: a parallel pipelined multimicro computer system for image creation"
Proc. of the 10 th symposium on Computer Architecture
SIGARCH, 1983, pp. 387 - 394

* [OME 88] - D. Ould Brahim et M. Mériaux
"Triangularisation de polygones quelconques"
Publication n° 56, USTLFA, Février 1988

* [OUL 88] - D. Ould Brahim
"Etude d'un nouveau modèle de primitive pour la description des images numériques: le macro-pixel"
Thèse de Docteur Ingénieur, Lille, Septembre 1988

* [PAV 82] - T. Pavlidis
"Algorithms for graphics and image processing"
Springer - Verlag, 1982

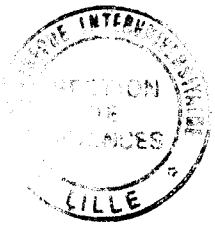
* [PEL 85] - M. Pelerin
"Synthèse d'images et parallélismes: algorithmes et architectures"
Thèse de Docteur de 3 ème cycle, Lille, Janvier 1985

* [PER 88] - B. Peroche et AL
"La synthèse d'images"
Hermès traité des nouvelles technologies, 1988

* [PHO 75] - B.T. Phong
"Illumination for computer generated pictures"
Communications ACM, Vol. 18, N° 6, Juin 1975

* [RED 84] - M. REDJIMI
"Etude et réalisation d'un système parallèle pour le traitement graphique"
Thèse de Docteur Ingénieur, Lille, Septembre 1984

* [TEX 87] - Texas Instruments
"TMS 34010 Users Guide"
Texas Instruments, 1987



* [ULL 83] - M.K. ULLNER
"Parallel machines for computer graphics"
Ph.D. Thesis - California Institute of Technology
Pasadena - California, 1983

ETUDE D'ARCHITECTURES PARALLELES

A BASE DE

PROCESSEURS GRAPHIQUES SPECIALISES

Les besoins de performances, en temps d'exécution et capacité de traitement, pour des applications graphiques, font évoluer les systèmes informatiques concernés vers soit des architectures parallèles soit des architectures spécialisées.

Ce travail a pour objectif de définir et d'étudier la parallélisation des processeurs graphiques spécialisés, en tirant avantage des deux aspects. On distingue trois grandes possibilités de parallélisme: découpage géométrique, découpage fonctionnel, et découpage par objet.

Pour chacun de ces découpages, une étude décrit les différentes solutions et compare leurs performances, afin de retenir les meilleures d'entres elles. L'étude prend en compte, à la fois, le parallélisme des processeurs et les fonctions spécifiques résultant de leur spécialisation. Il apparait que le niveau de parallélisme varie suivant le mode de partage.

Mots - clés:

Synthèse d'images, parallélisme, algorithmique graphique, processeur graphique spécialisé, partage objet, partage géométrique.