THÈSE

présentée à

L'UNIVERSITÉ DES SCIENCES ET TECHNIQUES DE LILLE

pour obtenir

LE TITRE DE DOCTEUR INGÉNIEUR

par

Michel MÉRIAUX

ÉTUDE ET RÉALISATION D'UN TERMINAL GRAPHIQUE COULEUR TRIDIMENSIONNEL FONCTIONNANT PAR TACHES

Thèse soutenue le 29 janvier 1979 devant la Commission d'Examen

Membres du Jury : MM.	P. BACCHUS	Président
	C. CARREZ	Examinateur
	J. LOSFELD	Examinateur
	M. LUCAS	Examinateur
	M. MELCHIOR	Examinateur
	V. CORDONNIER	Rapporteur

UNIVERSITE DES SCIENCES
ET TECHNIQUES DE LILLE

01.02.1976

DOYENS HONORAIRES de l'Ancienne Faculté des Sciences

MM. R. DEFRETIN, H. LEFEBVRE, M. PARREAU.

PROFESSEURS HONORAIRES des Anciennes Facultés de Droit et Sciences Economiques, des Sciences et des Lettres

M. ARNOULT, Mme BEAUJEU, MM. BROCHARD, CHAPPELON, CHAUDRON, CORDONNIER, CORSIN, DEHEUVELS, DEHORS, DION, FAUVEL, FLEURY, P. GERMAIN, HEIM DE BALSAC, HOCQUETTE, KAMPE DE FERIET, KOUGANOFF, LAMOTTE, LASSERRE, LELONG, Mme LELONG, MM. LHOMME, LIEBAERT, MARTINOT-LAGARDE, MAZET, MICHEL, NORMANT, PEREZ, ROIG, ROSEAU, ROUBINE, ROUELLE, SAVART, WATERLOT, WIEMAN, ZAMANSKI.

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

MM. R. DEFRETIN, M. PARREAU.

PRESIDENT DE L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE

M. J. LOMBARD.

PROFESSEURS TITULAIRES

M.	BACCHUS Pierre	Astronomie
M.	BEAUFILS Jean-Pierre	Chimie Physique
M.	BECART Maurice	Physique Atomique et Moléculaire
M.	BILLARD Jean	Physique du Solide
M.	BIAYS Pierre	Géographie
M.	BONNEMAN Pierre	Chimie Appliquée
M.	BONNOT Ernest	Biologie Végétale
M.	BONTE Antoine	Géologie Appliquée
M.	BOUGHON Pierre	Algèbre
M.	BOURIQUET Robert	Biologie Végétale
M.	CELET Paul	Géologie Générale
M.	CONSTANT Eugène	Electronique
M.	DECUYPER Marcel	Géométrie
M.	DELATTRE Charles	Géologie Générale
M.	DELHAYE Michel	Chimie Physique
M.	DERCOURT Michel	Géologie Générale
M.	DURCHON Maurice	Biologie Expérimentale
M.	FAURE Robert	Mécanique
M.	FOURET René	Physique du Solide
M.	GABILLARD Robert	Electronique
M.	GLACET Charles	Chimie Organique
M.	GONTIER Gérard	Mécanique
M.	GRUSON Laurent	Algèbre
M.	GUILLAUME Jean	Microbiologie
M.	HEUBEL Joseph	Chimie Minérale
M.	LABLACHE-COMBIER Alain	Chimie Organique
M.	LANSRAUX Guy	Physique Atomique et Moléculaire
M.	LAVEINE Jean-Pierre	Paléontologie
M.	LEBRUN André	Electronique
M.	LEHMANN Daniel	Géométrie

Mme	LENOBLE Jacqueline	Physique Atomique et Moléculaire
M.	LINDER Robert	Biologie et Physiologie Végétales
M.	LOMBARD Jacques	Sociologie
M.	LOUCHEUX Claude	Chimie Physique
M.	LUCQUIN Michel	Chimie Physique
M.	MAILLET Pierre	Sciences Economiques
M.	MONTARIOL Frédéric	Chimie Appliquée
M.	MONTREUIL Jean	Biochimie
M.	PARREAU Michel	Analyse
M.	POUZET Pierre	Analyse Numérique
M.	PROUVOST Jean	Minéralogie
M.	SALMER Georges	Electronique
M.	SCHILTZ René	Physique Atomique et Moléculaire
Mme	SCHWARTZ Marie-Hélène	Géométrie
M.	SEGUIER Guy	Electrotechnique
M.	TILLIEU Jacques	Physique Théorique
M.	TRIDOT Gabriel	Chimie Appliquée
M.	VIDAL Pierre	Automatique
M.	VIVIER Emile	Biologie Cellulaire
M.	WERTHEIMER Raymond	Physique Atomique et Moléculaire
M.	ZEYTOUNIAN Radyadour	Mécanique

PROFESSEURS SANS CHAIRE

M.	BELLET Jean	Physique Atomique et Moléculaire
M.	BODARD Marcel	Biologie Végétale
M.	BOILLET Pierre	Physique Atomique et Moléculaire
M.	BOILLY Bénoni	Biologie Animale
M.	BRIDOUX Michel	Chimie Physique
M.	CAPURON Alfred	Biologie Animale
M.	CORTOIS Jean	Physique Nucléaire et Corpusculaire
M.	DEBOURSE Jean-Pierre	Gestion des entreprises
M.	DEPREZ Gilbert	Physique Théorique
M.	DEVRAINNE Pierre	Chimie Minérale
M.	GOUDMAND Pierre	Chimie Physique
M.	GUILBAULT Pierre	Physiologie Animale
M.	LACOSTE Louis	Biologie Végétale
Mme	LEHMANN Josiane	Analyse
M.	LENTACKER Firmin	Géographie
M.	LOUAGE Francis	Electronique
Mle	MARQUET Simone	Probabilités
M.	MIGEON Michel	Chimie Physique
M.	MONTEL Marc	Physique du Solide
M.	PANET Marius	Electrotechnique
M.	RACZY Ladislas	Electronique
M.	ROUSSEAU Jean-Paul	Physiologie Animale
M.	SLIWA Henri	Chimie Organique

MAITRES DE CONFERENCES (et chargés d'Enseignement)

M.	ADAM Michel	Sciences Economiques
M.	ANTOINE Philippe	Analyse
M.	BART André	Biologie Animale
M.	BEGUIN Paul	Mécanique
M.	BKOUCHE Rudolphe	Algèbre
M.	BONNELLE Jean-Pierre	Chimie
M.	BONNEMAIN Jean-Louis	Biologie Végétale
M.	BOSCQ Denis	Probabilités
M.	BREZINSKI Claude	Analyse Numérique
M.	BRUYELLE Pierre	Géographie

.../

M. CARREZ Christian	Informatique
M. CORDONNIER Vincent	Informatique
M. COQUERY Jean-Marie	Psycho-Physiologie
Mle DACHARRY Monique	Géographie
M. DEBENEST Jean	Sciences Economiques
M. DEBRABANT Pierre	Géologie Appliquée
M. DE PARIS Jean-Claude	Mathématiques
M. DHAINAUT André	Biologie Animale
M. DELAUNAY Jean-Claude	Sciences Economiques
M. DERIEUX Jean-Claude	Microbiologie
M. DOUKHAN Jean-Claude	Physique du Solide
M. DUBOIS Henri	Physique
M. DYMENT Arthur	Mécanique
M. ESCAIG Bertrand	Physique du Solide
Me EVRARD Micheline	Chimie Appliquée
M. FONTAINE Jacques-Marie	Electronique
M. FOURNET Bernard	Biochimie
M. FORELICH Daniel	Chimie Physique
M. GAMBLIN André	Géographie
M. GOBLOT Rémi	Algèbre
M. GOSSELIN Gabriel	Sociologie
M. GRANELLE Jean-Jacques	Sciences Economiques
M. GUILLAUME Henri	Sciences Economiques
M. HECTOR Joseph	Géométrie
M. JACOB Gérard	Informatique
M. JOURNEL Gérard	Physique Atomique et Moléculaire
Mle KOSMAN Yvette	Géométrie
M. KREMBEL Jean	Biochimie
M. LAURENT François	Automatique
Mle LEGRAND Denise	Algèbre
Mle LEGRAND Solange	Algèbre
M. LEROY Jean-Marie	Chimie Appliquée
M. LEROY Yves	Electronique
M. LHENAFF René	Géographie
M. LOCQUENEUX Robert	Physique Théorique
M. LOUCHET Pierre	Sciences de l'Education
M. MACKE Bruno	Physique
M. MAHIEU Jean-Marie	Physique Atomique et Moléculaire
Me N'GUYEN VAN CHI Régine	Géographie
M. MAIZIERES Christian	Automatique
M. MALAUSSENA Jean-Louis	Sciences Economiques
M. MESSELYN Jean	Physique Atomique et Moléculaire
M. MONTUELLE Bernard	Biologique Appliquée
M. NICOLE Jacques	Chimie Appliquée
M. PAQUET Jacques	Géologie Générale
M. PARSY Fernand	Mécanique
M. PECQUE Marcel	Chimie Physique
M. PERROT Pierre	Chimie Appliquée
M. PERTUZON Emile	Physiologie Animale
M. PONSOLLE Louis	Chimie Physique
M. POVY Lucien	Automatique
M. RICHARD Alain	Biologie
M. ROGALSKI Marc	Analyse
M. ROY Jean-Claude	Psycho-Physiologie
M. SIMON Michel	Sociologie
M. SOMME Jean	Géographie
Mle SPIK Geneviève	Biochimie
M. STANKIEWICZ François	Sciences Economiques
M. STEEN Jean-Pierre	Informatique

.../...

M.	THERY Pierre	Electronique
M.	TOULOTTE Jean-Marc	Automatique
M.	TREANTON Jean-René	Sociologie
M.	VANDORPE Bernard	Chimie Minérale
M.	VILLETTE Michel	Mécanique
M.	WALLART Francis	Chimie
M.	WERNIER Georges	Informatique
M.	WATERLOT Michel	Géologie Générale
Mme	ZINN-JUSTIN Nicole	Algèbre

Je tiens à remercier Monsieur le Professeur P. BACCHUS, qui a bien voulu me faire l'honneur de présider le jury de cette thèse.

Je suis très reconnaissant envers Monsieur le Professeur V. CORDONNIER, qui m'a proposé ce travail et m'a guidé de ses conseils.

Je remercie Monsieur le Professeur C. CARREZ et Monsieur J. LOSFELD, Maître de Conférences, qui se sont intéressés à ce travail et ont accepté de participer au jury.

Je remercie particulièrement Monsieur M. LUCAS, Maître de Conférences à l'Université de Grenoble, et Monsieur MELCHIOR de la CIT-Alcatel, d'avoir accepté de participer a ce jury.

Que les membres du Laboratoire d'Architecture des Machines, en particulier D. HOUVIEZ et P. BEHAGUE, ainsi que ceux du Laboratoire d'Optique Atmosphérique, soient assurés de ma gratitude pour m'avoir supporté et aidé de leurs conseils durant plus de deux années.

Je tiens également à remercier Mademoiselle B. FIEVET, ainsi que Madame et Monsieur P. DEBOCK, qui ont assuré avec diligence, gentillesse, et compétence la réalisation matérielle de ce document.

A QUOI PEUT BIEN SERVIR

UN LIVRE SANS IMAGE ?

LEWIS CARROLL

(ALICE AU PAYS DES MERVEILLES)

SOMMAIRE

-:-:-:-:-:-:-:-:-:-

-:-:-:-:-:

3. LE SYSTEME PROPOSE

-:-:-:-:-:

4. FAISABILITE DU PROCESSEUR GRAPHIQUE

-:-:-:-:-:

5. APPLICATIONS ET VOIES DE RECHERCHE

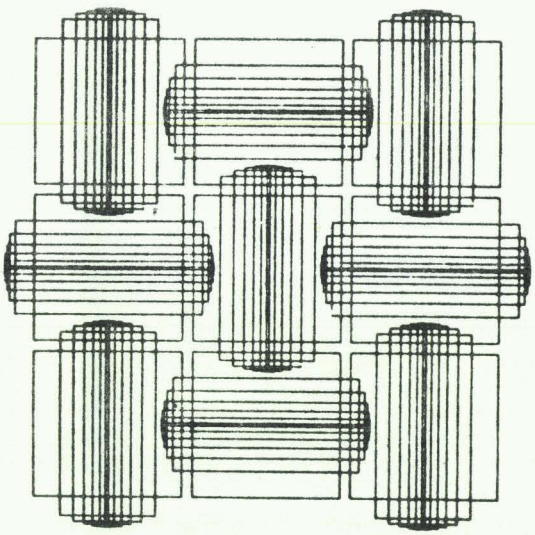
INTRODUCTION

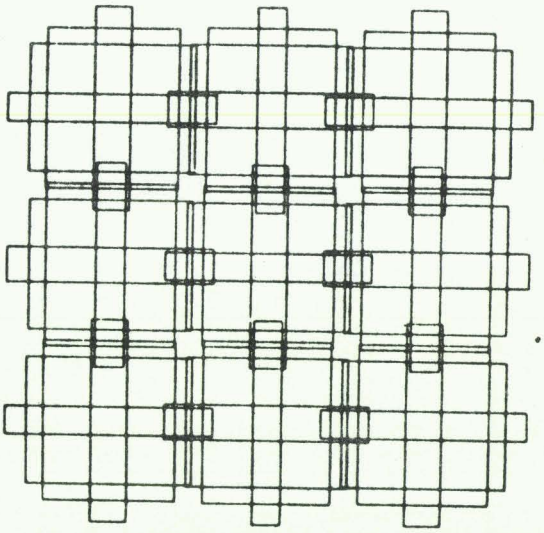
L'idée de cette étude naquit au début de l'année 1977. Elle résulte de plusieurs constatations : d'une part le dispositif de communication visuelle le plus commun est le téléviseur ; d'autre part, les systèmes de visualisation disponibles montrent certaines limitations, qui peuvent être la faiblesse des performances, le coût exagéré, ou l'inadaptation à certains problèmes.

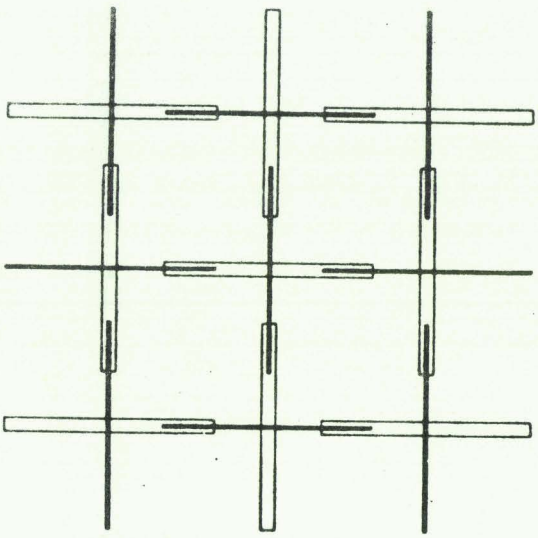
Nous avons donc cherché à réaliser un système graphique disposant d'un téléviseur comme élément final d'affichage, mais éliminant les défauts liés à un tel choix, autant que faire se peut. En particulier la quantité d'information manipulée nous a amené à analyser divers procédés de codage et à en définir un qui, d'une part donne un taux de compression d'information satisfaisant, d'autre part conserve le contenu sémantique de l'image, à l'aide d'une description par taches de celle-ci.

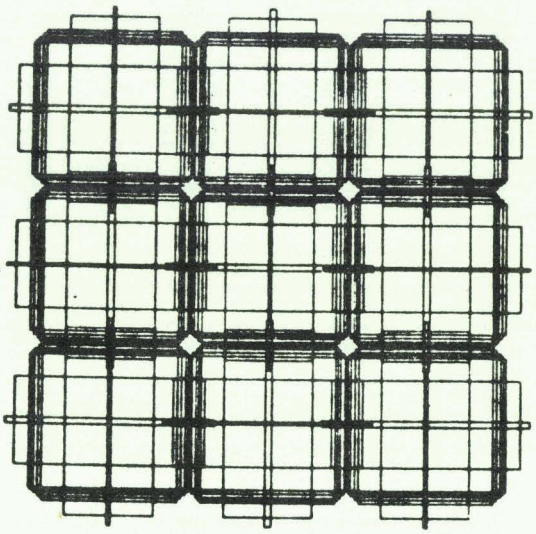
En cours d'étude sont apparues certaines contraintes et certaines caractéristiques spécifiques, qui ont quelque peu fait dévier de sa route le projet initial : l'accent a été notamment mis sur l'aspect tridimensionnel du fonctionnement, et sur l'impossibilité technologique de réaliser le système idéal ainsi défini. Le compromis retenu pour la réalisation conserve une part des objectifs initiaux mais rapporte en traitement séquentiel ce que la technologie disponible et le coût n'ont pas permis de réaliser en totale simultanéité.

Ce travail s'articule donc en cinq chapitres, correspondant à une succession chronologique : le premier, rappelle brièvement ce qu'est la visualisation graphique, et présente quelques systèmes existants. Le second s'attache à préciser la notion d'"information image", notamment en ce qui concerne les procédés de codage ou de transformation couramment utilisés. Compte-tenu de contraintes initiales, le troisième s'efforce de définir un système graphique, et en particulier de justifier le choix du moyen de codage utilisé. Le quatrième étudie la faisabilité de ce qui est sinon le coeur, du moins le cerveau du terminal, c'est-à-dire le processeur graphique ; un certain nombre de solutions sont proposées, dont une, celle retenue, est détaillée en annexe. Le dernier chapitre utilise certaines des caractéristiques du système apparues a posteriori pour ouvrir quelques voies de recherches et mettre en lumière les applications privilégiées par le système proposé.









CHAPITRE 1

-:-:-:-:-:-:-:-:-:-:-

GENERALITES

1 GENERALITES

1-1 L'"infographie" ([Mor 76])

Nous ne pouvons aborder un domaine aussi vaste que la visualisation graphique d'informations (ou "infographie", pour employer un néologisme récent) sans tenter de faire le point des recherches et de l'existant. Cette analyse aura également l'avantage de situer notre problème dans ce domaine.

De quoi parlons-nous ? Si nous analysons le terme "infographie", nous en déduisons immédiatement qu'il s'agit de la visualisation d'informations sous formes de dessins ou d'images. Une telle définition englobe aussi bien le texte imprimé que les photographies de famille. Nous ne nous interresserons pour notre part qu'aux informations susceptibles d'un traitement informatique ; notons au passage que ce critère est tout-à-fait subjectif et ne rejette ni le texte, ni la photo de famille. Il n'est pas dans notre propos ici de justifier l'importance grandissante qu'a l'image dans la communication et en particulier en informatique. Retenons simplement ceci : "Le principal intérêt de l'image est la présentation synthétique qu'elle permet de faire, présentation utilisant l'outil d'analyse puissant qu'est le sens de la vision. Son contenu peut être facilement compréhensible par de larges classes de personnes ayant une culture analogue". ([Luc 78])

Nous appelerons donc "système graphique" un système informatique traitant et finalement visualisant sous forme d'images (au sens large) des informations de toutes natures.

1-2 Caractérisation d'un système graphique. ([Luc 76])

L'architecture fonctionnelle d'un tel système peut être décrite de la façon suivante :

- un élément final d'affichage de l'image : le "terminal"
- des éléments de calcul : les "processeurs"
- des dispositifs de communication avec l'utilisateur

- un ensemble de programmes destinés à faciliter l'utilisation du système : le "logiciel"
- un ensemble de programmes écrits pour résoudre le problème particulier de l'utilisateur

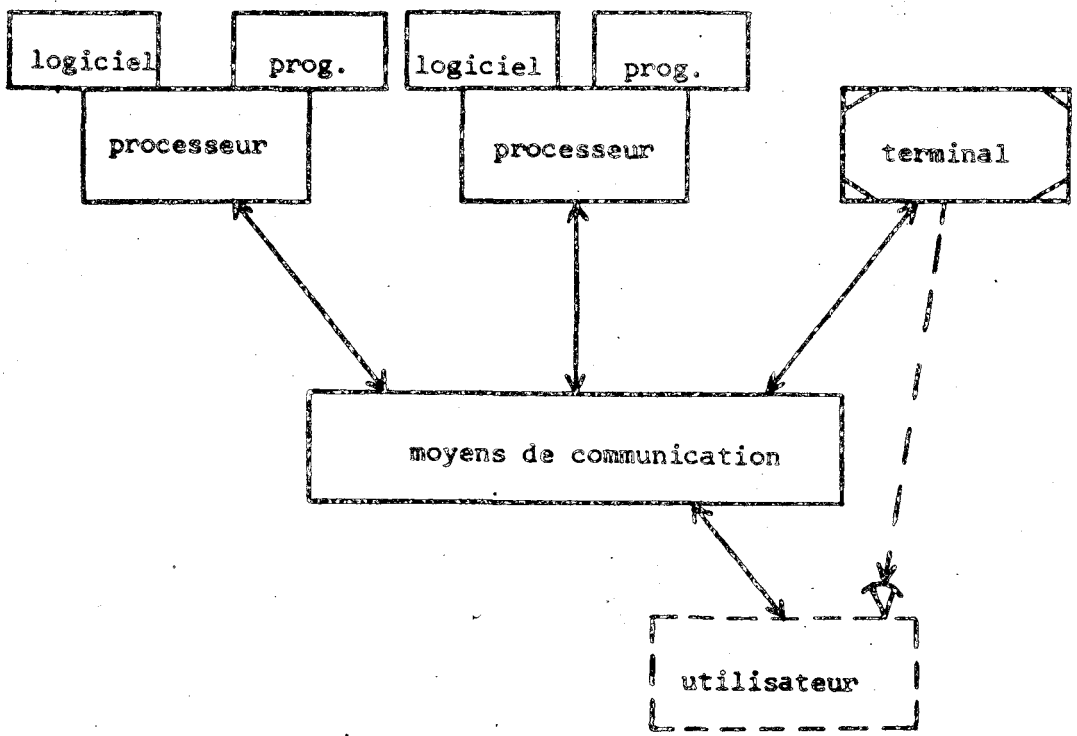


Figure 1.a : Système graphique, architecture fonctionnelle.

1-2-1 Le terminal.

Nous pouvons y distinguer d'une part l'écran, d'autre part le processeur graphique. L'écran détermine la qualité de l'image finalement affichée, à savoir sa taille, sa persistance, sa couleur, sa finesse. La technologie actuelle permet de distinguer les écrans à mémoire, des écrans à rafraîchissement, c'est-à-dire où l'inscription de l'image doit être refaite périodiquement. Les premiers sont monochromes ou ne disposent que de quelques couleurs ; ils ne sont en général effaçables que globalement, mais leur capacité élevée de mémorisation les rend simples d'emploi. Les écrans à rafraîchissement, quant à eux, ont en théorie au moins l'avantage d'être très interactifs (en effet, pour modifier une portion d'image, il suffit de la réinscrire différemment.) ; ils permettent de nombreuses couleurs.

Il s'agit des tubes de télévision, mais aussi des écrans à laser. Les écrans à plasma sembleraient allier les avantages des deux systèmes.

En ce qui concerne le processeur graphique, qui est en fait l'interface entre le système vu par l'usager et l'écran, sa complexité peut être très variable : il est chargé d'interpréter les ordres qu'il reçoit des éléments de calcul et de les transformer en commandes directes sur l'écran. Sa rapidité, son "intelligence" sont déterminantes dans la caractérisation de la puissance d'un terminal.

1-2-2 Les éléments de calcul.

Nous pouvons y distinguer le (ou les) processeur (s) local (aux) au terminal, du (ou des) processeur (s) éloigné (s), c'est-à-dire le plus souvent relié par voie de télétransmission. Le premier effectue toutes les tâches simples et renvoie au second les plus complexes, c'est-à-dire trop longues ou nécessitant trop d'espace mémoire. Le processeur éloigné peut d'ailleurs être un réseau, ce qui met clairement en évidence ce problème essentiel qu'est le temps de transmission des données sur les lignes téléphoniques ; ce temps dépend étroitement du codage des informations graphiques, que nous étudierons en détail .

1-2-3 Les dispositifs de communication.

L'utilisateur peut et doit avoir à sa disposition un certain nombre de moyens de communication avec le système. Ce sont

- soit des moyens classiques d'entrée-sortie, à savoir télétypes, claviers alpha-numériques, imprimantes, lecteurs de rubans perforés, de cassettes, etc... sur lesquels nous ne nous étendrons pas.

- soit des moyens spécifiques de l'infographie, et qui peuvent être :

  . le photostyle ou la boule roulante, qui permettent de repérer un point sur l'écran, et d'y dessiner (en association avec un marqueur)

  . la table à digitaliser, qui retransmet au processeur les coordonnées du "crayon" avec lequel on dessine

- l'appareil photographique, qui est un moyen simple et rapide, notamment grâce aux systèmes de type Polaroïd; de sortie durable sur papier d'une image.

Ces moyens spécifiques se développent et sont de plus en plus sophistiqués. Nous pourrions cependant les classer en

- moyens de sortie d'image : appareil photographique, imprimante, table traçante
- moyens d'entrée/sortie de coordonnées de points : boule, table à digitaliser, curseur, photostyle etc...

Ces derniers sont une indication importante quant à l'interactivité du système.

1-2-4 Le logiciel.

Son contenu et ses performances dépendent à la fois de la complexité du terminal et des désirs de l'utilisateur. Il peut contenir par exemple un processeur de langage (par exemple BASIC étendu ou FORTRAN 3D, ou encore un langage spécifique), ou bien encore un logiciel permettant la visualisation d'objets de l'espace à trois dimensions. Il peut aussi mettre à la disposition de l'usager des opérateurs graphiques (rotation, translation,...), des ordres élémentaires (changer de couleur, etc...).

1-2-5 Les programmes de l'utilisateur.

Ces programmes, spécifiques d'une utilisation particulière, sont parfois développés par le fabricant du système graphique. Mais, le plus souvent, c'est l'utilisateur lui-même qui doit écrire ces programmes en fonction de ce que le logiciel lui fournit. Il est à regretter l'existence d'une grande diversité de logiciels et de programmes utilisateurs ; cependant une certaine standardisation commence à apparaître.

1-3 Quelques systèmes existants.

Nous allons ici décrire quatre systèmes graphiques, correspondant à des utilisations assez distinctes.

1-3-1 TEKTRONIX : Système de calcul graphique 4051 ([TEK]).

Ce système inclut ou peut inclure les éléments suivants :

- un écran à mémoire monochrome de 1024 × 780 points, de 28 cm de diagonale (soit environ 20 points au mm)
- un processeur graphique interprétant soit des ordres alphanumériques, soit des ordres graphiques en mode vecteur
- un processeur local (8K à 32 Koctets ; extension de 300 K sur cassette magnétique) supportant un Basic étendu (manipulations de chaînes de caractères ; fonctions mathématiques ; opérateurs WINDOW, VIEWPORT, ROTATE). Ce processeur est connectable à une unité de disques, à une unité de cartouches magnétiques, à une imprimante, ou encore à un ordinateur
- des moyens d'entrée-sortie graphique, à savoir un "manche à balai", une table traçante, ou une unité de reprographie (table traçante dont le fonctionnement s'apparente à celui d'un télécopieur)
- des logiciels divers (auto-enseignement ; tracés de courbes, histogrammes ou fonctions ;...)
- des bibliothèques de programmes pour applications spéciales (statistiques, mathématiques, électronique)

La version de base comporte l'écran, le processeur graphique, le processeur local (8Koctets + un lecteur de cartouches) et une interface standard (IEEE 488).

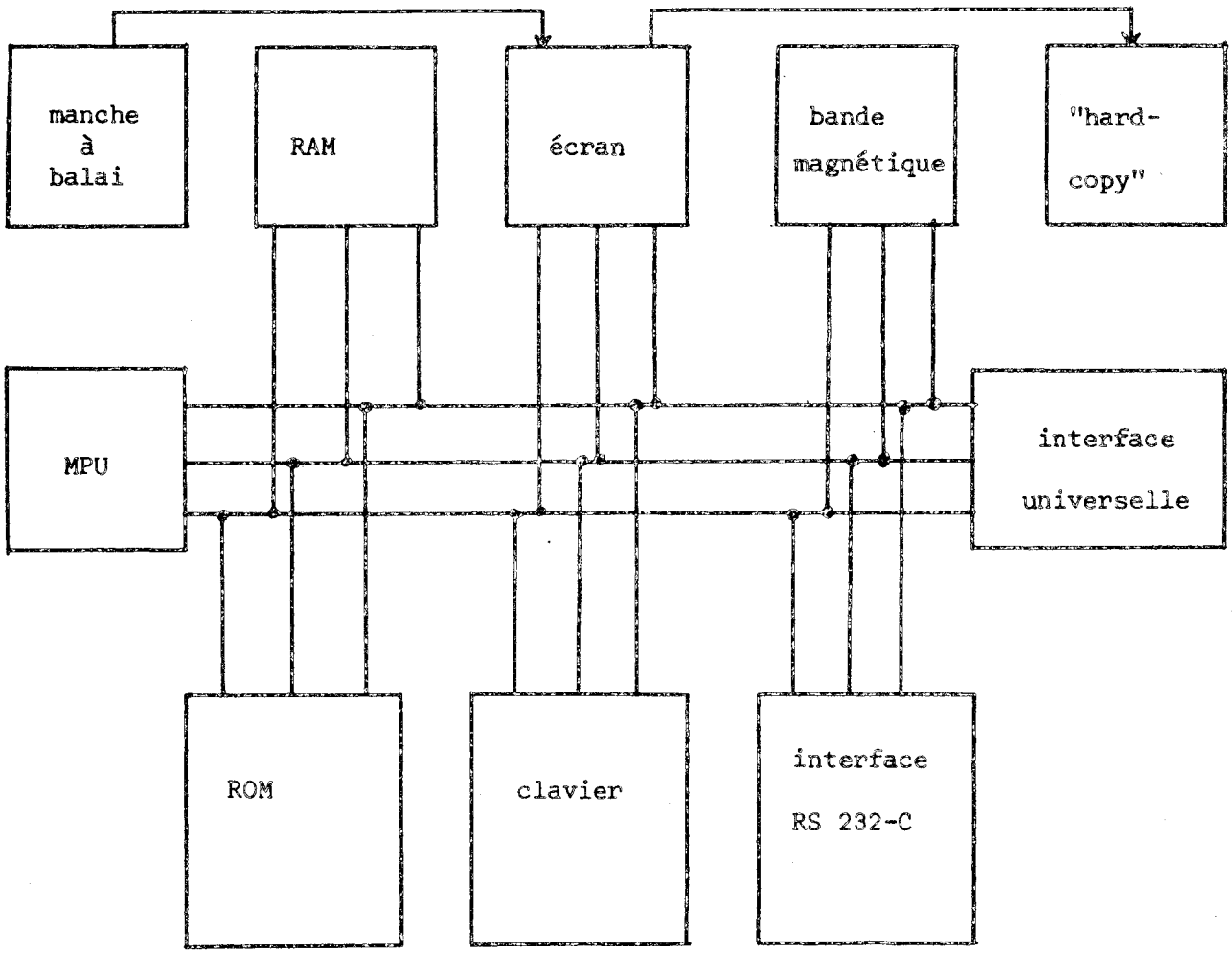


Figure 1.b : Tektronix 4051. Exemple de configuration.

Ce système est plutôt destiné à la création et à la représentation de tableaux, diagrammes, fonctions ou schémas complexes. Il n'est pas adapté par contre à l'affichage, ni a fortiori au traitement d'images d'origine photographique, à cause de sa lenteur d'affichage et de sa monochromie (affichage en tout ou rien). De plus tout effacement partiel est impossible : il faut tout effacer, et réinscrire à nouveau.

1-3-2 CIT-Alcatel : Terminal de Traitement et de présentation d'Images : TRIM. ([CIT])

Comme son nom l'indique, ce système a pour vocation première le traitement et la visualisation d'images numérisées.

- l'écran peut être soit un téléviseur standard, soit un moniteur professionnel permettant l'affichage de 512 lignes de 512 points, chaque point étant codé sur 1 à 8 bits (soit 256 couleurs disponibles)
- le processeur graphique comporte essentiellement une (ou des) mémoire(s) de rafraîchissement de l'écran et un (ou des) contrôleur(s) mémoire pouvant :
  - . écrire des données issues du processeur
  - . relire des données pour les lui transférer
  - . relire la mémoire pour rafraîchir l'écran. Cette relecture est sélective et permet des effets de zoom, défilement ou juxtaposition d'images.

On peut inclure également les "contrôleurs vidéo", qui réalisent certaines fonctions plus évoluées :

- sélection des données affichées
- utilisation de seuils pour les données numériques
- dilatation de la tranche des données
- translation de l'origine
- génération de "fausses couleurs" à l'aide d'une table

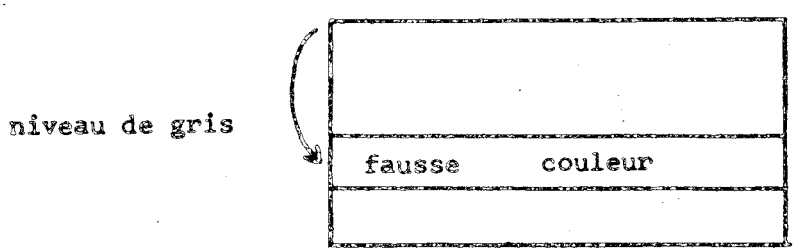


Figure 1.c : Table de "Fausses couleurs".

- le système dispose d'un certain nombre de moyens de dialogue, dont :
  - . une boule roulante associée à un marqueur sur l'écran
  - . un panneau général de commandes
- il n'y a pas de processeur local : le terminal doit être connecté à un ordinateur (appelé "pilote"), qui effectue les tâches plus évoluées

- on peut disposer de logiciels effectuant la gestion du terminal, et la réalisation de traitements et de manipulations d'images (en FORTRAN)

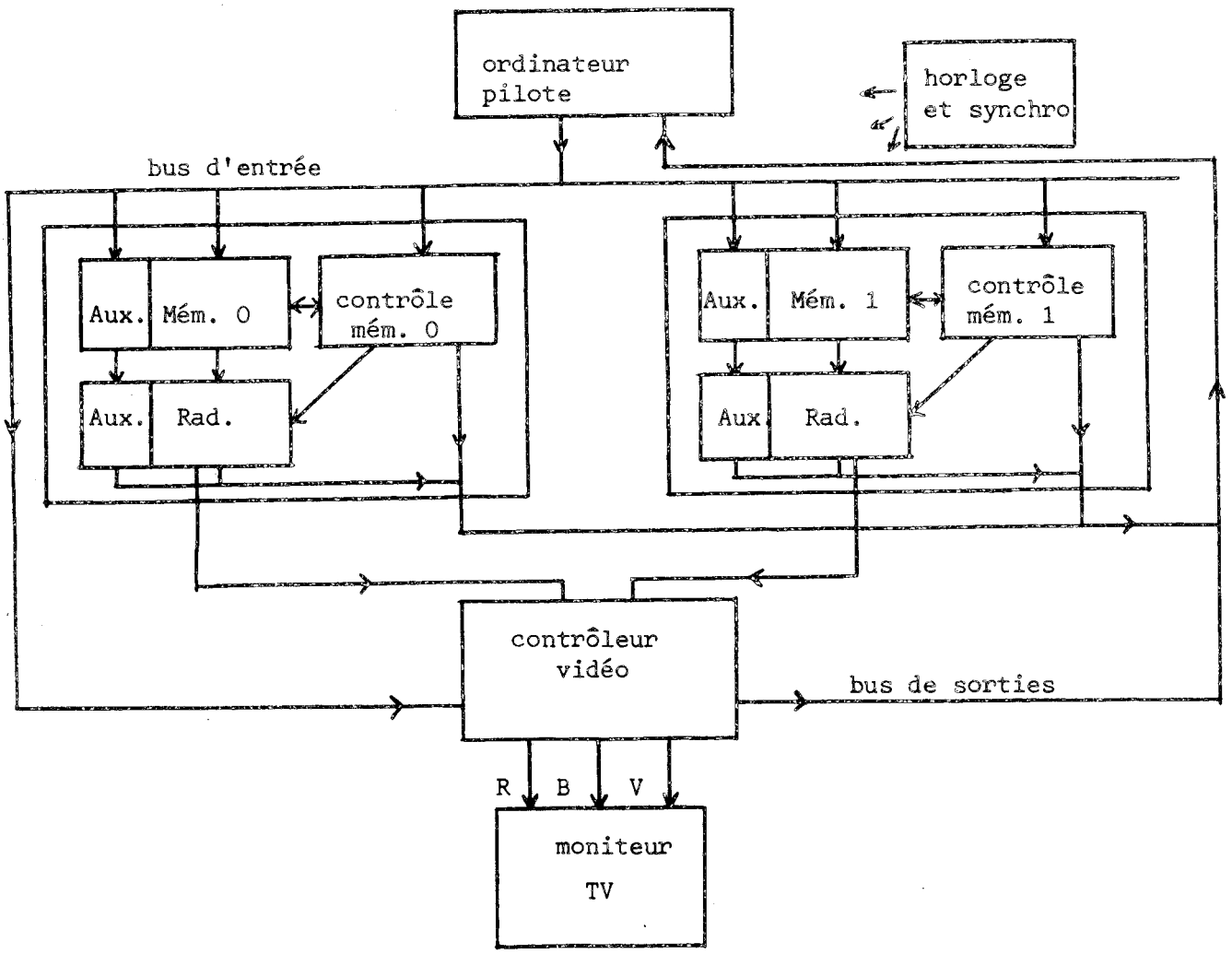


Figure 1.d : TRIM. (système Multitrim, exemple)

Les applications visées sont :

- recherches sur images (en astronomie, géographie, géophysique, météorologie, physique des matériaux, etc...)
- recherche médicale et bio-médicale, à partir de photographies, radiographies, thermographies, tomographies, scintigraphies...

- créations graphiques et picturales
- observation et étude des sols et paysages

A la différence du système précédent, ce système est destiné à la visualisation en couleurs d'images numérisées, mais il est absolument inadapté à l'affichage de dessins, graphes, courbes ou schémas. Il est également important de constater la nécessité de disposer d'un ordinateur "pilote", car le fonctionnement en mode local est très limité.

1-3-3 SEIN : Imagerie Couleur Numérique PERICOLOR ([Per]).

Nous restons ici dans le domaine de la visualisation d'images numérisées, avec un système à la fois plus simple, mais plus autonome que le TRIM. Il comporte :

- un écran, qui est un téléviseur couleur standard, permettant d'afficher une image de 64 × 64 jusqu'à 256 × 256 points, chaque point étant codé sur 3 à 12 bits (4095 couleurs disponibles) ; la visualisation peut être faite en vraies ou en "fausses" couleurs. (Dans ce cas on dispose de 16 couleurs parmi 255)
- un processeur graphique incluant
  - . la mémoire de rafraîchissement de l'écran, dont la taille peut aller de 4K à 64K mots de 4 à 17 bits
  - . un générateur de caractères et de symboles, permettant l'affichage de 1800 caractères sur l'écran. L'utilisateur a le choix entre 64 caractères standards et 64 symboles graphiques, et ce parmi 7 couleurs en positif, négatif ou clignotant
  - . une interface d'entrée de données et de paramètres, qui permet d'écrire dans la mémoire, de préciser les paramètres de visualisation en fausses couleurs, de préciser les dimensions de l'image, de charger le générateur de caractères et d'assurer le couplage avec le processeur local.
- le processeur local (micro-ordinateur INTEL 8080) qui rend le système autonome. Il peut effectuer un certain nombre de traitements élémentaires sur les données, en mémoire de rafraîchissement, à savoir : lissage, intégration, addition ou soustraction d'images, etc...

Ce processeur dispose de 1 Koctets de mémoire vraie et de 1 à 8 Koctets de mémoire reprogrammable. Il peut être couplé à un clavier ou à une imprimante

- des moyens de dialogue (photostyle ou boule roulante)

Le système peut être connecté à un ordinateur pour effectuer les traitements les plus complexes.

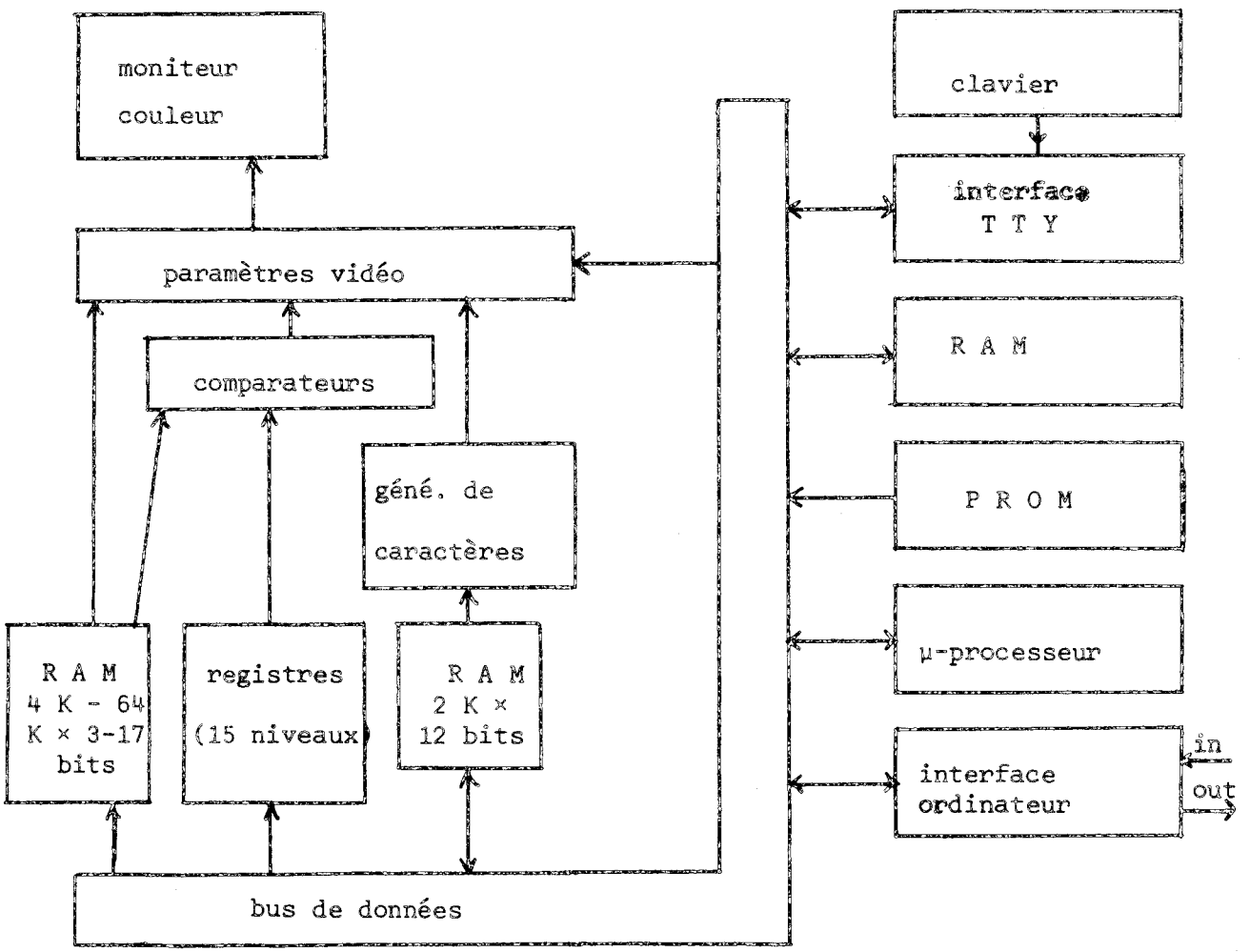


Figure 1.e : Le système PERICOLOR

Les applications visées sont voisines de celles du système TRIM, à savoir :

- télédétection (météorologie, exploration des ressources par satellites, imagerie infra-rouge, contrôle de pollution)
- recherche fondamentale (physique nucléaire)

- biologie
- centralisation de mesures
- contrôle industriel
- arts graphiques (télévision, cinéma, textile)

1-3-4 Université de Paris VIII : COLORIX. ([Col 76])

Ce système, un peu différent des précédents, est né des préocupations du groupe "Art et Informatique" de l'Université de Vincennes. Il a été construit à partir d'un certain nombre de contraintes, à savoir :

- avoir un système permettant de traiter facilement de l'information colorée
- disposer d'un grand nombre de couleurs, dans le but d'étudier la perception visuelle
- pouvoir modifier l'image rapidement (pour faire de l'animation)
- programmer facilement et utiliser de manière interactive en temps réel
- pouvoir connecter facilement le système sur n'importe quel ordinateur
- pouvoir le transporter facilement
- avoir un système "peu coûteux"

Ces contraintes particulières ont amené à la réalisation d'un système composé de :

- un téléviseur couleur affichant 57 rangées de 71 carrés (ou "pixels"), chacun occupant environ 5 mm sur 5 mm. La couleur est codée sur 12 bits (4 par couleur primaire), ce qui donne un choix très vaste de 4096 teintes
- un processeur graphique, essentiellement composé de :
  - . un générateur de signaux de synchronisation
  - . une mémoire de rafraîchissement (4K x 12 bits)
  - . un contrôleur de dialogue terminal/ordinateur

Il n'y a pas dans cette version de processeur local ; l'ordinateur connecté écrit directement dans la mémoire de rafraîchissement.

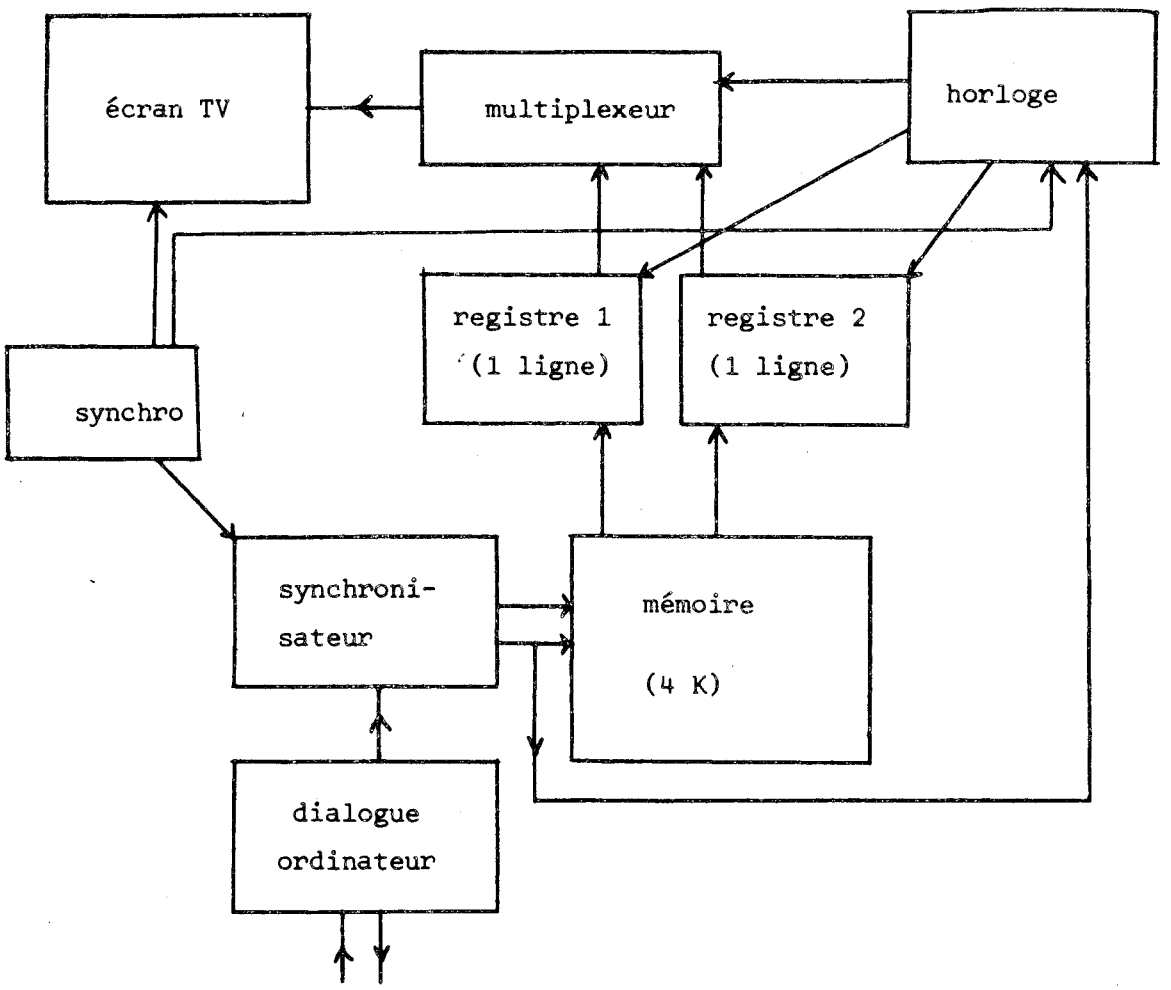


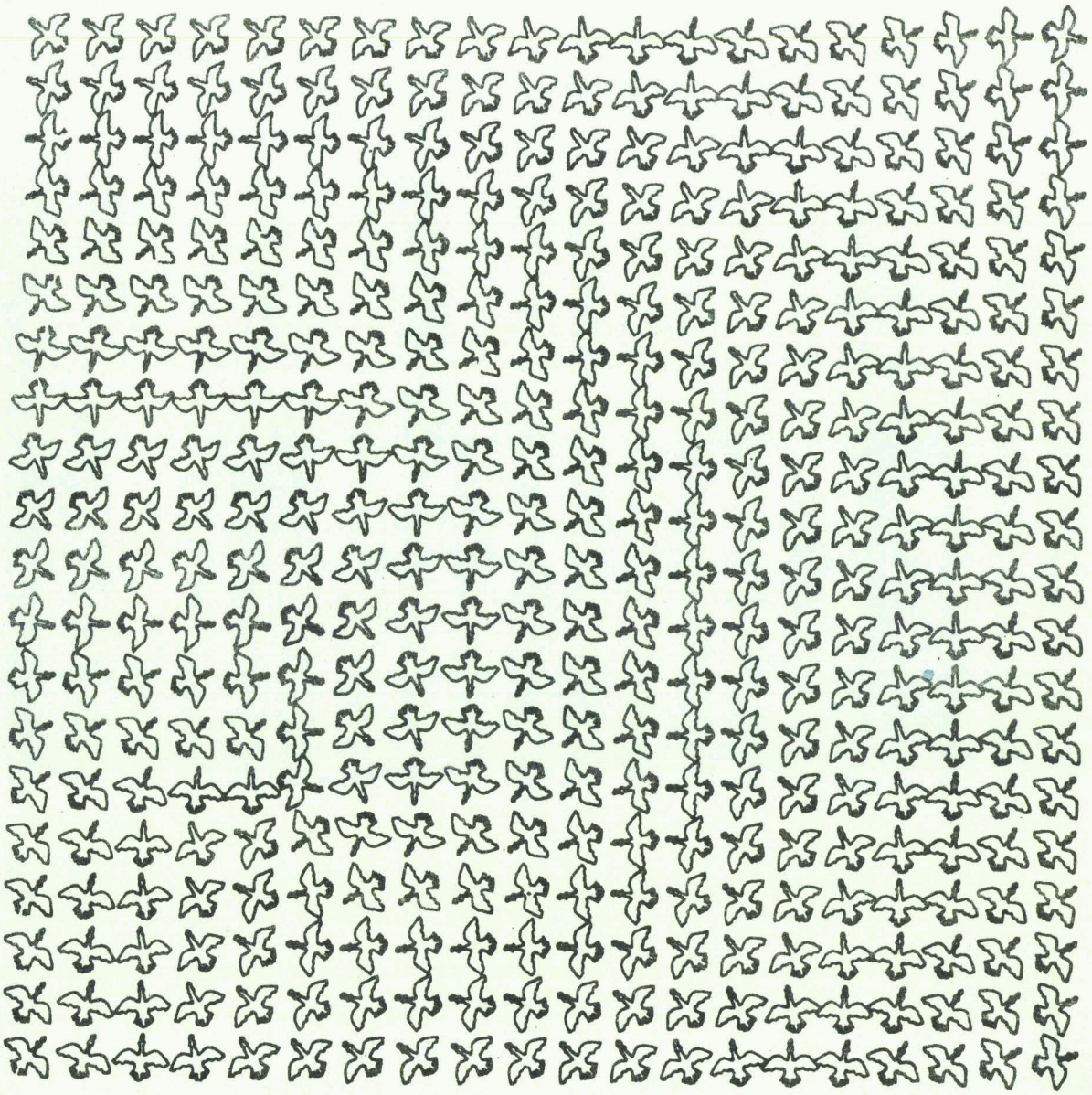
Figure I.f : COLORIX.

Ce système répond assez bien aux contraintes initiales, il est envisagé de l'étendre, par augmentation de la mémoire, adjonction d'un micro-processeur local, etc...

1-3-5 Conclusion

Ces quatre systèmes, que nous avons décrits suivant l'architecture générale définie plus haut, différent en de nombreux points : nous voyons clairement que les préoccupations d'un statisticien utilisant le Textronix 4051 sont différentes de celles d'un artiste utilisant Colorix ou de celles d'un géophysicien utilisant le Péricolor.

D'une part les images dont ils s'occupent sont de natures diverses, d'autre part l'utilisation qui en est faite n'est pas la même. Nous allons donc être amené à définir le cahier des charges de notre système relativement à ces deux paramètres, la nature de l'image, et le type d'utilisation qui en est faite.



CHAPITRE 2

-:-:-:-:-:-:-:-:-:-:-:-:-

L'INFORMATION IMAGE

2 L'INFORMATION IMAGE

Nous cherchons ici à définir et à analyser l'objet sur lequel nous nous proposons de travailler, à savoir l'image que l'utilisateur d'un système graphique manipule. Pour ce faire, il nous faudra analyser comment il crée cette image et ce qu'il en fait. Il apparaîtra alors que la description codée est sinon indispensable, du moins intéressante à de nombreux points de vue. La description d'un certain nombre de procédés de codage permettra par la suite d'orienter le choix vers un codage particulier, compte tenu des contraintes matérielles que nous nous serons fixées.

2-1 Définitions.

On peut définir une "image" comme un ensemble structuré d'informations, susceptible, après affichage sur un écran approprié, d'avoir un sens pour l'oeil humain. Cependant nous limitons l'espace d'observation final à une surface rectangulaire représentable par une matrice de n * m points, chaque point (ou "pixel") ayant une couleur c(x,y).

Nous supposons également que l'image est discrétisée et quantifiée dans tout le système graphique ; en d'autres termes, lorsque l'image provient du monde réel, le problème des capteurs est supposé traité et résolu préalablement ; même hypothèse quant à l'intégration lors de la restitution finale (table traçante, écran de télévision, oeil humain, etc...).

Enfin nous ferons la distinction entre

- ce que l'on voit sur l'écran (un ensemble de points lumineux)
- la représentation dans le système graphique (liste d'affichage, contenu de la mémoire de rafraîchissement ou analogue)
- le monde physique réel, dans le cas où il est à l'origine de l'image

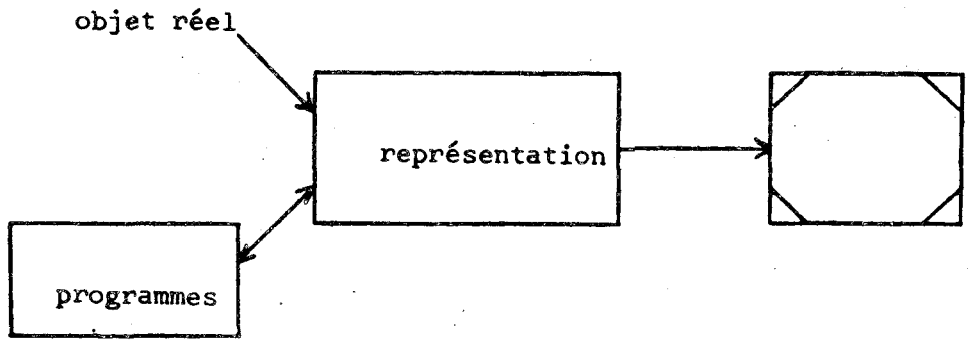


Figure 2.a

2-2 Différents types d'images. ([Gon 76])

Il est important de distinguer les images suivant d'une part leur mode de création, d'autre part l'utilisation que l'on en fait. En effet, les performances demandées au système graphique dépendent étroitement de ces données.

2-2-1 L'image photographique.

2-2-1-1 Origine

On regroupe sous cette dénomination toutes les images provenant d'un système optique ou électronique, et justifiant un traitement informatique. Ce sont par exemple les photographies aériennes ou de satellites, ou encore les radiographies médicales.

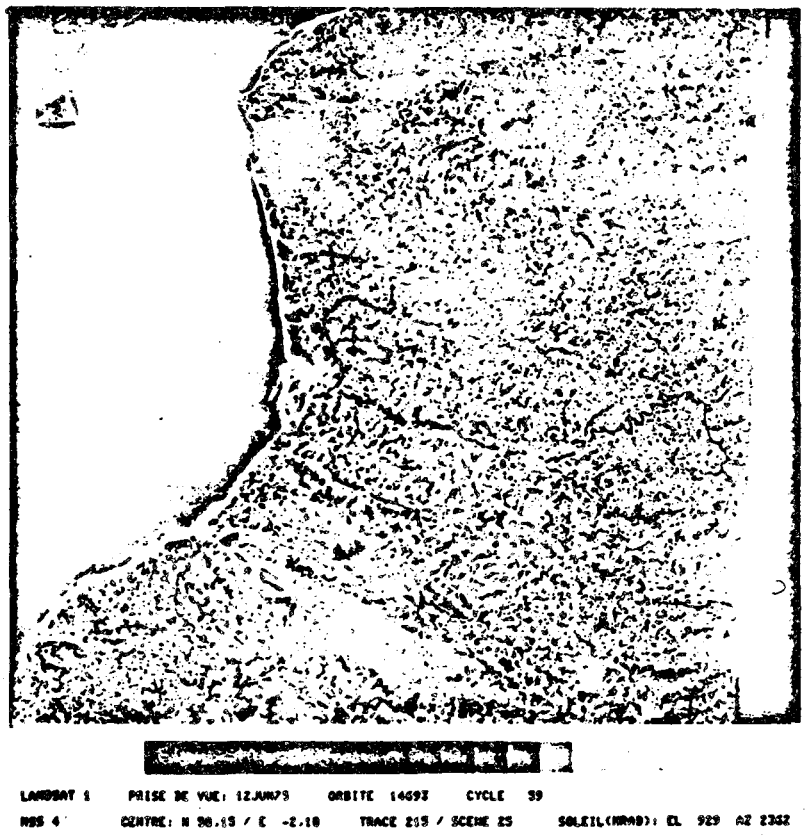


Figure 2.b : La terre vue d'un satellite. (image LANDSAT)

2-2-1-2 Description.

Ces images sont discrétisées et quantifiées sous forme d'une matrice de points ; chaque point est affecté d'un certain niveau (dit "de gris"). Cette matrice est souvent traitée de façon séquentielle, ligne après ligne, comme par exemple pour le stockage sur bandes magnétiques ou lors de l'affichage sur un écran de télévision.

2-2-1-3 Caractéristiques.

Une telle image est plane, sauf dans les cas, très exceptionnels, de vues stéréoscopiques ou d'hologrammes. D'autre part elle est complètement définie dès l'entrée dans le système graphique : en effet tout traitement n'enrichira pas le contenu de l'image, mais le modifiera ; l'image initiale est la plus complexe, celles obtenues a posteriori, calculées à partir de l'image initiale, sont plus simples, même si elles mettent en évidence ce que l'utilisateur y cherche.

Ainsi un filtrage de bruit sur une photographie diminue la quantité d'information présente dans l'image ; néanmoins le contenu sémantique (apparition de zones polluées sur une photographie aérienne) ou syntaxique (analyse de scènes) est évidemment et heureusement augmenté.

Retenons aussi qu'une telle image est toujours représentée ou représentable par une matrice de n x m points, chaque point étant codé sur 2^q niveaux de gris, soit une quantité d'information égale à n x m x q bits. Ainsi une image en provenance d'un satellite (NASA) nous donne 2 000 lignes de 2 000 points environ, codés sur un octet (soit 256 niveaux de gris) et impose donc une capacité de 32 Mégabits.

2-2-2 L'image programmée.

2-2-2-1 Origine.

C'est une image dont l'utilisateur a programmé (non interactivement) la génération. Ce sont par exemple les résultats graphiques d'un calcul, ou encore la génération automatique de dessins.

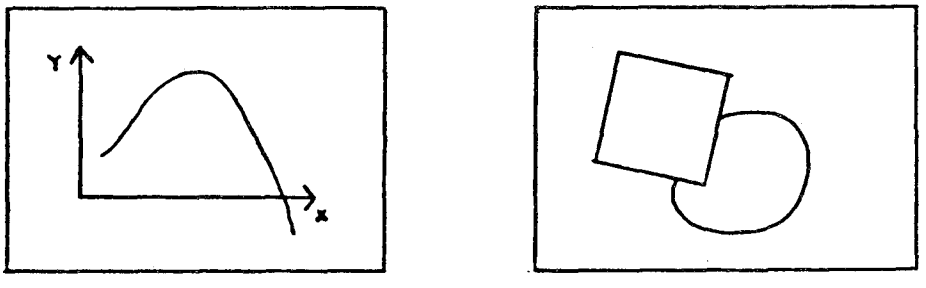


Figure 2.c Exemples d'images programmées.

2-2-2-2 Description.

Elle peut revêtir des formes très variées, depuis la description matricielle jusqu'à la description par grammaire d'éléments primitifs.

Citons notamment les listes de coordonnées souvent utilisées sur les tables traçantes ou les écrans à mémoire.

2-2-2-3 Caractéristiques.

Ici encore, l'image, bien que progressivement créée, est rigidement définie. Il n'est donc pas nécessaire de disposer d'un codage permettant la réciprocité, c'est dire de retrouver à partir de l'image affichée les éléments de l'image sourcegénérée par le programme. L'image est toujours composée d'un certain nombre d'objets, qui peuvent être soit des lignes (graphiques, schémas, etc...) soit des taches colorées (dessin animé), soit, dans le cas fréquent où l'on travaille dans l'espace à trois dimensions, des volumes. Notons que dans ce dernier cas, le passage de l'espace de travail à l'espace d'affichage à deux dimensions est un problème essentiel, qui doit être pris en charge soit par programme, soit par le processeur graphique.

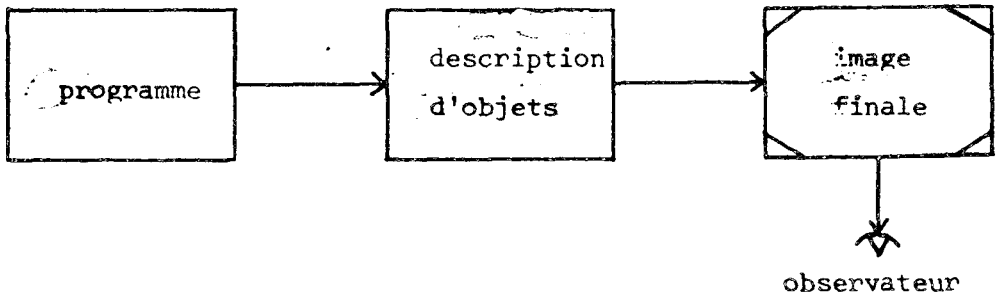


Figure 2.d : Image programmée.

2-2-3 L'image interactive.

2-2-3-1 Origine.

L'utilisateur la crée progressivement à l'aide des moyens d'entrée (clavier alphanumérique, boule roulante, table à numériser) et des procédures informatiques mises à son service. Il peut, de façon générale, désigner, modifier, retrancher, ajouter tout ou partie de l'image.

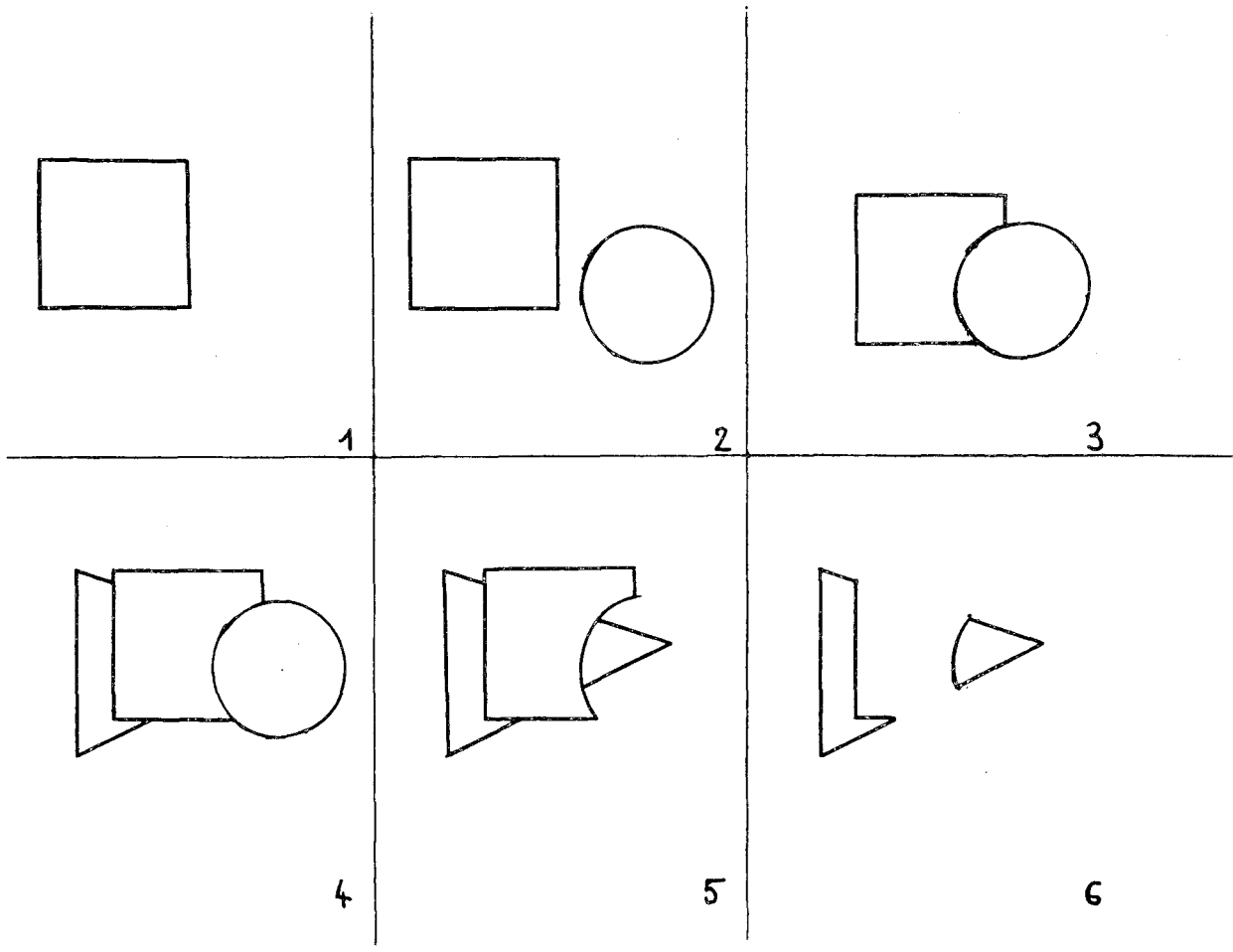


Figure 2.c : exemple.

2-2-3-2 Description.

Elle est ici aussi très variable mais le plus souvent les moyens d'entrée impliquent une description par objets, chaque objet étant défini par la donnée de son contour ou d'éléments permettant sa construction (objets primitifs, description géométrique, etc...).

2-2-3-3 Caractéristiques.

Toutes les actions sur les objets ou sur l'image complète sont ici essentielles, et devraient pouvoir se faire en temps réel ;

la création d'un dessin animé exigerait par exemple un temps de réponse du système de l'ordre du dixième de seconde (ordre de grandeur de la rémanence visuelle). Si de plus on travaille dans l'espace, la méthode de résolution des lignes ou surfaces cachées a une grande importance.

Il est ici indispensable de pouvoir remonter de l'image affichée aux objets connus du système, c'est-à-dire de pouvoir définir l'appartenance d'un point à un objet.

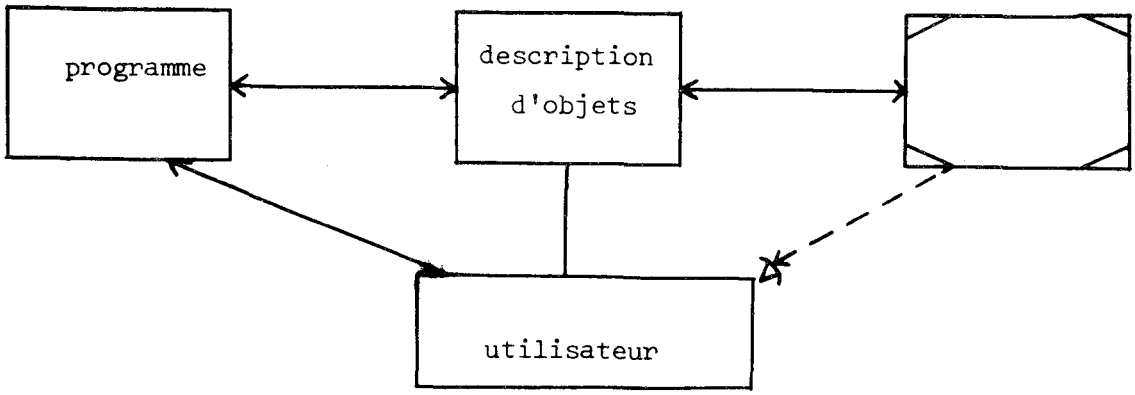


Figure 2.f : Image interactive

Même si les distinctions que nous avons établies peuvent paraître parfois artificielles, car on peut trouver des images pouvant relever de plusieurs types, nous retiendrons une séparation assez nette entre l'image photographique, décrite sous forme matricielle, et l'image créée par programme, ou interactivement. La première implique des traitements globaux importants, souvent longs, donc relevant d'un ordinateur puissant ; la seconde nécessite moins de calculs, moins d'espace de travail, mais un système graphique assez évolué pour supporter les interactions, le passage de 3 à 2 dimensions, les actions localisées sur des éléments de l'image. Si la première impose l'affichage sur un écran à hautes performances (couleur, résolution), la seconde pourrait se satisfaire d'un écran à balayage cavalier, à mémoire ; mais l'utilisation d'un écran couleur à rafraîchissement devrait permettre d'une part d'utiliser l'information couleur, d'autre part de diminuer le temps de réponse du système.

2-3 Nécessité du codage d'image.

Un certain nombre d'arguments militent en faveur du codage de l'image, et nous allons les détailler ici.

2-3-1 Espace de stockage.

Lorsque l'écran ne peut stocker lui-même d'image, ou même dès qu'il faut travailler sur l'image, il faut la conserver en mémoire. Sous forme matricielle (m x n points et q bits par point) il faut un espace considérable (m x n x q bits). En outre, si cette mémoire doit rafraîchir l'écran, elle doit être rapide, donc coûteuse.

Exemples : 1) image de satellite NASA : m = n = 2 000 ; q = 8 (256 niveaux) : il faut donc 4 M octets.

2) image télévision en couleurs réduite à n = m = 256 et q = 8 (256 couleurs) : il faut 64 K octets.

(note : ces images seront prises comme références dans ce qui suit).

S'il faut rafraîchir l'écran 50 fois par seconde, le temps d'accès moyen à chaque octet serait

- dans le premier cas : $\bar{t}_1 \# \frac{1}{50 \times 4 \times 10^6} = 5$ ns (!)
- dans le second : $\bar{t}_2 \# \frac{1}{50 \times 64 \times 10^3} \# 300$ ns

Dans le premier cas, il conviendrait de n'afficher l'image que par morceaux, de façon à avoir des temps d'accès technologiquement réalisables.
Dans le second cas, le plus réaliste, le temps d'accès réel serait plutôt de l'ordre de 150 ns (à cause des temps morts dus aux retours lignes et trames) et est tout-à-fait réalisable.

2-3-2 Transmission en ligne de l'image.

Il est fréquemment nécessaire de transmettre l'information graphique entre l'ordinateur et l'élément terminal, ou entre des ordinateurs (cas d'un réseau) ; ceci se faisant par voie téléphonique, le temps de transmission peut devenir prohibitif. Si v est la vitesse de transmission de la ligne (en bauds), q la quantité d'information à transmettre et t le temps théorique de transmission, on a

$$t = \frac{q}{v} \text{ (en secondes)}$$

cas	v bauds / q	100	300	1 200	4 800	9 600
(1)	32 M	≠100 H	≠ 32 H	≠ 8 H	≠ 2 H	≠ 1 H
(2)	512 K	1h20	25mn	7mn	1mn 40 s	50 sec
(3)	64 K	10mn	3mn	50 sec	12 sec	6 sec

(1) image de satellite : q = 32 Mégabits
(2) image TV : q≃512 Kilobits
(3) image TV codée avec un taux de compression de 8 : q = 64K bits

Compte tenu de l'impatience habituelle de l'utilisateur, l'avantage d'un codage est évident.

2-3-3 Information utile.

Soit une image-type de télévision, unicolore. Sa description matricielle impose une quantité d'information de 512 Kilobits. Or la seule information réellement présente est la couleur de l'image, soit 8 bits. Il y a donc un rapport de l'ordre de 10^5 entre la quantité d'information que donnerait la description matricielle et celle dont nous avons vraiment besoin.

Nous pouvons ainsi imaginer que, quelle que soit l'image, il existe une quantité minimale d'information, que nous appelerons information utile (nous pourrions l'appeler "entropie") ; cette information, inaccessible en général, permettrait la reconstruction exacte de l'image originale sous forme matricielle. Si le procédé de codage tente d'approcher ce minimum idéal, il est clair que :

1) l'information contenue dans le code sera inférieure à celle contenu dans la description matricielle (sauf cas particuliers)
2) cette information variera comme l'information utile.

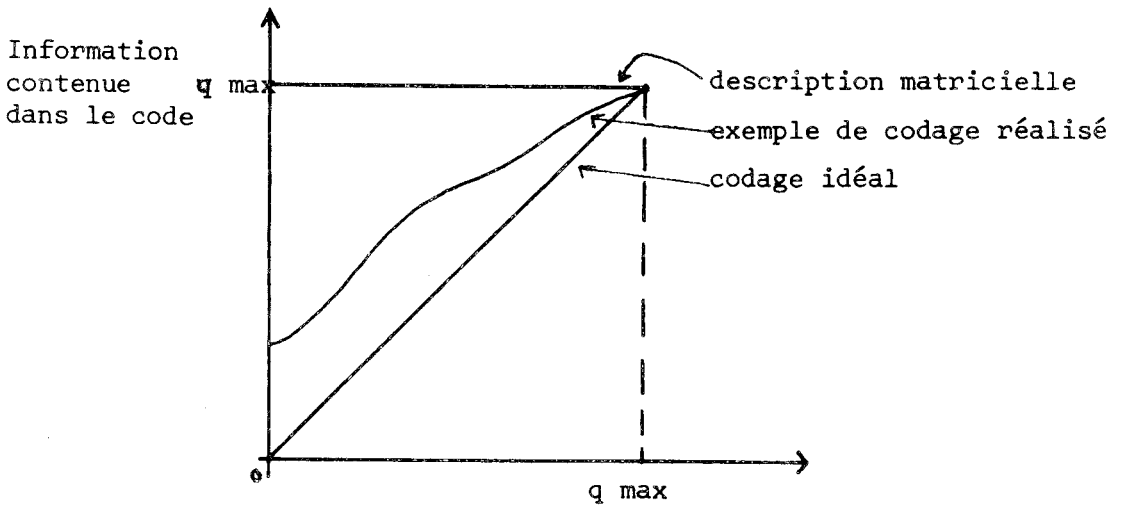


Figure 2.g : Codage et information utile.

Remarque : Comme toujours en théorie de l'information, cette analyse est valable dans un contexte de travail donné : plus le contexte est riche, plus l'information utile, donc celle contenue dans le code utilisé, est faible. Ainsi dans un contexte graphique prenant en compte des cercles, un cercle est défini par

c = cercle (centre, rayon)

Si le contexte ne connaît pas l'objet cercle, et ne connaît par exemple que les vecteurs, il sera défini par une suite de coordonnées

$c = \{(x_1,y_1), (x_2,y_2), \ldots\ldots, (x_1,y_1)\}$

Nous voyons donc l'importance que peut avoir l'"intelligence" du processeur graphique ou du logiciel disponible.

2-3-4 Transfert d'information nécessité par une action.

Nous appelons "action" tout phénomène susceptible de modifier l'information contenue dans l'image. Elle doit toujours se réaliser le plus vite possible. Deux moyens sont utilisables conjointement :

- rendre toutes les actions réalisables localement, ce qui implique un codage facilitant leur implémentation, le processeur local étant en général peu puissant
- minimiser les transferts entre le processeur éloigné et le terminal, en ne transmettant que des informations synthétiques

En voici deux exemples :

1) soit une image blanche que nous désirons verte ; sans codage, il faut transmettre : "en (1,1), afficher 'vert' ; en (1,2), afficher 'vert' ;.....etc".

Si le rectangle est connu du système (par 2 sommets opposés), on aurait :
"afficher 'vert' dans le rectangle [(1,1),(n,m)]"

2) soit un cercle C à translater de $(\Delta x, \Delta y)$
- "cercle" et "translater" sont inconnus
  "effacer $\{(x_1,y_1),(x_2,y_2),\ldots,(x_1,y_1)\}$; afficher $\{(x_1+\Delta x,y_1+\Delta y),(x_2+\Delta x,y_2+\Delta y),\ldots,(x_1+\Delta x,y_1+\Delta y)\}$
- "cercle" est connu, mais non "translater
  "effacer cercle $((x_o,y_o),R)$; afficher cercle $((x_o+\Delta x, y_o+\Delta y\ R)$"
- "cercle" et " translater" sont connus
  "translater cercle $((x_o,y_o), R)$ de $(\Delta x,\Delta y)$"
- si l'objet c = cercle $((x_o,y_o), R)$ est de plus connu, il vient "translater c de $(\Delta x,\Delta y)$"

2-4 Eléments de comparaison et d'évaluation.

Ces arguments devant suffire à justifier l'importance du codage et l'influence du choix de contexte dans lequel il est effectué, nous allons analyser plusieurs méthode de codage. Mais il convient de définir auparavant quelques critères qui permettront de guider un choix ultérieur ; nous en avons établi une liste aussi exhautive que possible.

2-4-1 Adéquation aux types d'images.

Une image existant sous une certaine forme initiale, il s'agit de déterminer si la méthode utilisée est ou non compatible avec cette forme. Ainsi une description matricielle est inadéquate pour décrire une figure géométrique, mais adaptée à la description d'une photographie aérienne.

2-4-2 Adéquation aux actions.

Lorsque l'utilisateur agit sur l'image, il veut que le résultat de cette action soit obtenu rapidement. Pour ce faire, il faut qu'elle soit exécutable facilement, si possible localement. Par exemple, la rotation d'un objet serait difficile dans un codage matriciel, mais facile si cet objet est connu en tant que tel.

Inversement un codage non matriciel impose le retour à une description matricielle pour pouvoir effectuer une transformée de Fourier globale (que l'on ne réalise par ailleurs généralement pas localement, et dont le temps d'exécution est assez souvent peu critique).

2-4-3 Taux de compression de l'information.

Nous avons vu qu'il était possible de définir une "information utile" contenue dans une image, mais qu'il était la plupart du temps difficile de la quantifier ou de l'extraire. Pour une image photographique, il s'avère même quasi impossible d'en donner un ordre de grandeur. En outre, elle varie selon l'observateur. Par contre, si nous supposons que nous manipulons une image nette, c'est-à-dire une image se réduisant sur l'élément terminal à une partition de zones bien limitées, caractérisées par une couleur (ou un algorithme de calcul de cette couleur), alors il paraît raisonnable d'affirmer que l'information utile varie dans le même sens que :

1) le nombre de zones de l'image

2) la longueur et la forme des contours de ces zones.

Ces paramètres ne sont pas exhaustifs, mais permettent de donner un moyen simple d'évaluation de l'information utile : nous définissons donc la "complexité" d'une image ainsi décrite comme étant le nombre total de points limites de zones, c'est-à-dire encore la somme des périmètres des zones. Cette grandeur servira dans la suite de moyen de mesure de l'information utile, même si elle est imparfaite : d'une part elle ne tient pas compte des couleurs ; d'autre part elle est tout-à-fait inadaptée si l'image est de nature périodique, répétitive ou géométrique.

Nous pouvons alors définir deux rapports de compression, qui nous permettront de comparer les méthodes de codage :

- T_1, utilisable surtout pour les images photographiques, défini par :

$$T_1 = \frac{q(\text{image sous forme matricielle})}{q\ (\text{image codée})}$$

- T_2, utilisable surtout pour les images nettes, égal à :

$$T_2 = \frac{q\ (\text{image codée})}{\text{complexité}}$$

2-4-4 Difficulté du codage

Le codage consiste à passer de l'image source, décrite sous une certaine forme (par exemple matricielle), à l'image codée. Il faut donc que ce processus soit le plus simple et le plus efficace possible (en temps et en espace). La comparaison se fera suivant le type d'image.

2-4-5 Difficulté du décodage

Il s'agit ici de traduire le code en éléments susceptibles d'être directement affichables sur l'écran. Le décodage dépend donc de la nature de l'écran ; aussi nous nous intéressons surtout aux terminaux à rafraîchissement. Dans un tel terminal, l'interprétation du code doit se faire à un rythme imposé ; il est donc important d'évaluer :

- la difficulté de l'algorithme
- la puissance de la machine le réalisant, à savoir sa taille, sa vitesse, ses limites

2-4-6 Mémoire de stockage associée

Si nous supposons nécessaire de stocker l'image codée, et c'est bien le cas dans un terminal à rafraîchissement, il y a lieu de préciser la taille, la structure, la vitesse et les modes d'accès de la mémoire réalisant ce stockage.

2-5 Description et comparaison de divers procédés.

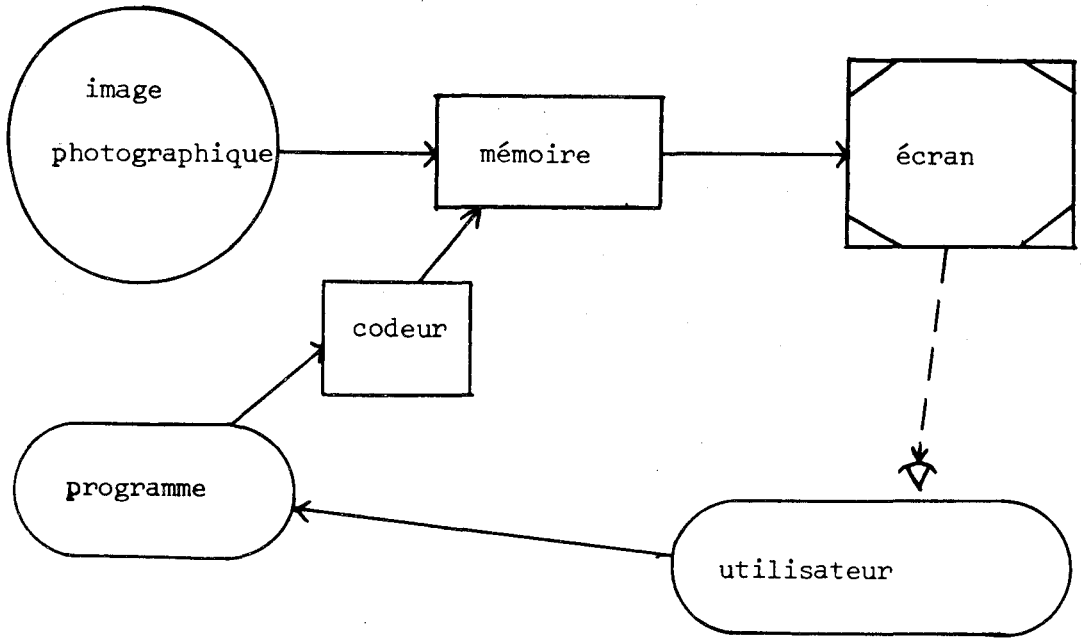
Nous allons ici étudier suivant les critères définis en 2-4 plusieurs procédés, dont certains sont orientés vers les images photographiques, d'autres plutôt vers le graphique.

2-5-1 La description matricielle

L'image est représentée sous la forme d'une matrice à n lignes et m colonnes, soit n x m éléments ; chaque élément (ou point) est codé sur q bits et on a donc 2^q niveaux possibles. Ceci est tout-à-fait adapté (et universellement utilisé) pour le traitement des images d'origine photographique, avec un taux de compression T_1 de 1.

Pour ce qui est des images des nature graphique, on a aussi $T_1 = 1$ mais cela n'a pas grande signification ; la quantité de code étant constante, nous en déduisons que pour de faibles complexités, $T_2 >> 1$. De plus, si le codage est simple et le décodage inexistant, les actions sur des éléments de l'image sont par contre malaisées, car la description ne les prend pas en compte. Par contre les actions globales (par exemple filtrage numérique) sont aisées.

Quant à la mémoire de stockage, elle doit être de forte capacité : nous avons vu qu'une image de satellite occupe environ 4 Méga-octets et une image de télévision 64 Kilo-octets. Si de plus la mémoire sert à rafraîchir un écran à balayage, le débit doit être élevé; de l'ordre de 8 Mégaoctets par seconde, ce qui implique des mémoires rapides, donc coûteuses. Nous verrons cependant que la technologie actuelle (1978) permet de conserver cette solution si la taille de l'image est faible ou si le coût a peu d'importance. Il est bien sûr certain que, dans un avenir assez proche, la technologie mettra à notre disposition des mémoires de forte capacité (plusieurs dizaines, voire centaines de kilo-octets en un seul boîtier), très rapides (les technologies I^2 L et ECl en particulier nous donnent des temps d'accès de l'ordre de quelques nano-secondes), peu chères (sans doute inférieures à 0,1 centime par bit), et de faible consommation. Dans le cas d'un dispositif à rafraîchissement, l'architecture serait :



2-5-2 Changement d'espace ([Zah 72], [And 76], [Roe 77])

On passe ici d'un image décrite de façon matricielle dans l'espace usuel

$$I = \{c(i,j) \mid 1 \leq i \leq n,\ 1 \leq j \leq m\}$$

à une matrice transformée dans un autre espace (dit "des fréquences", par analogie avec la transformée de Fourier de l'Electronique)

$$I^* = \{d(k,l) \mid 1 \leq i \leq k,\ o \leq l \leq p\}$$

à l'aide d'un calcul analytique pouvant être par exemple

- la transformation de Fourier bi-linéaire :

$$d(k,l) = \sum_{j=1}^{m} e^{2i\Pi lj} \left[\sum_{j=1}^{n} e^{2i\Pi ki} \; c(i,j) \right]$$

- la transformation de Fourier spatiale.

$$d(k,l) = \sum_{j=1}^{m} \sum_{i=1}^{n} e^{2i\Pi(lj+ki)} \; c(i,j)$$

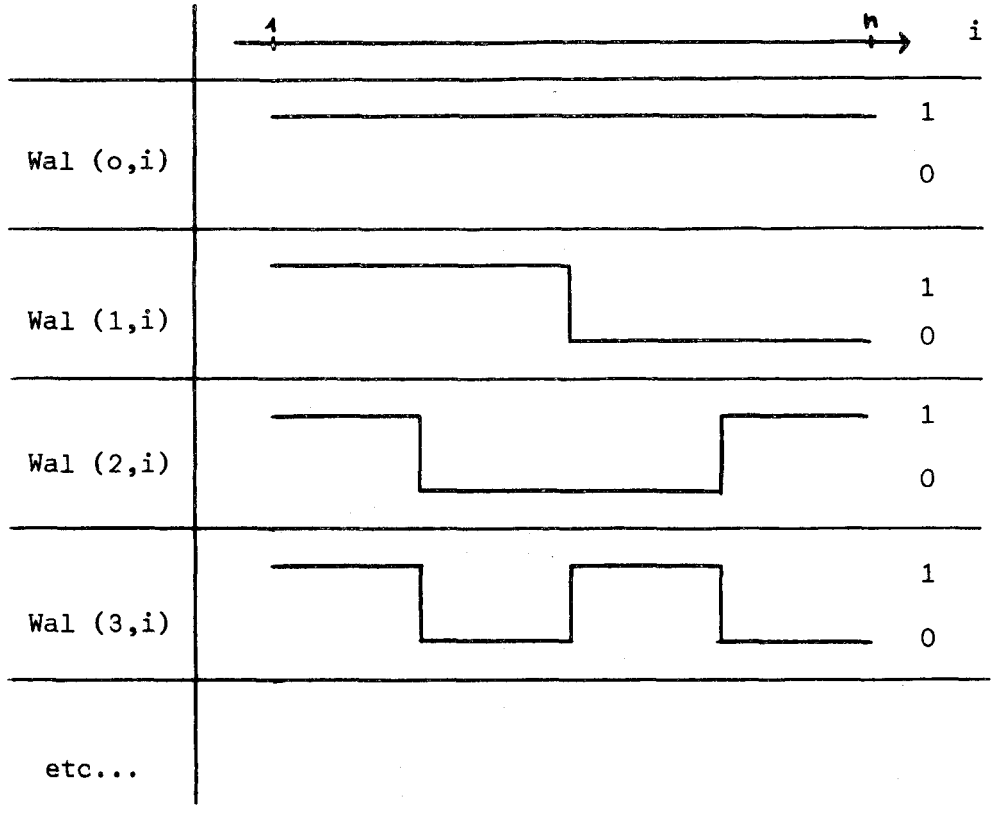
- transformation de Walsch-Hadamard

(à une dimension)

si $f(i)_n$ est l'image et F(u) sa transformée, on à

$$F(u) = \sum_{i=1}^{n} Wal\ (u,i)\ f(i)$$

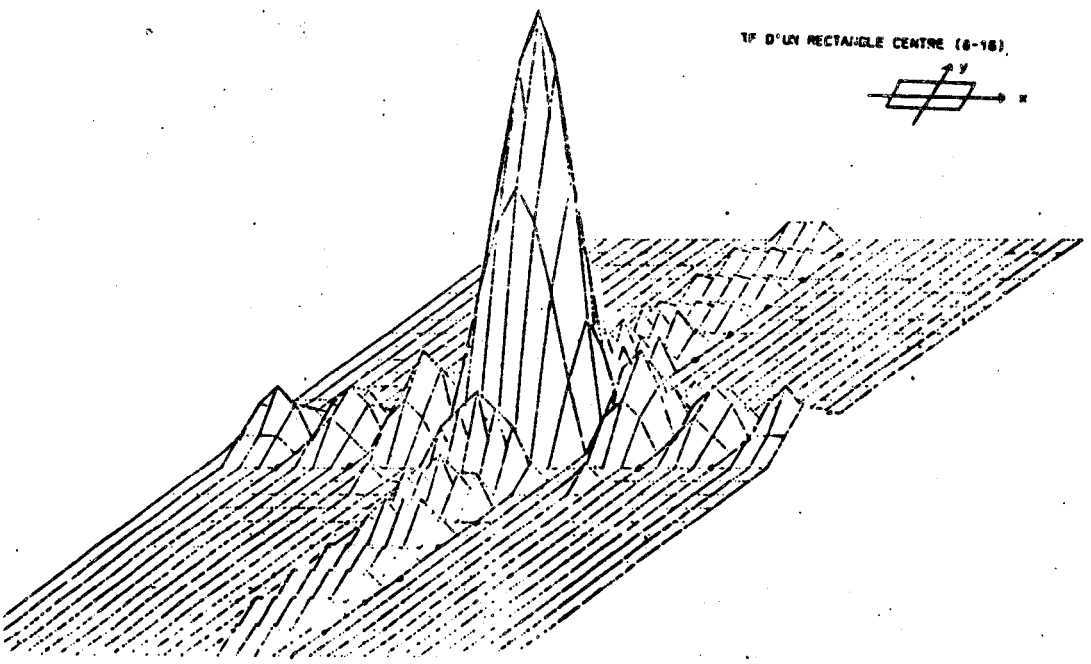
où $Wal^{i=1}(u,i)$ est la fonction de Walsch de la variable i, d'indice u, qui a l'allure suivante :



- la transformation de Haar, etc...

Ces méthodes sont uniquement applicables à une image préexistant sous forme matricielle. Le codage et le décodage (c'est-à-dire la transformation inverse) sont malaisés : par exemple, une transformation de Fourier spatiale effectuée sans précaution demande m x n multiplications et m n -1 additions complexes, soit environ 130 000 opérations complexes. En outre le résultat du codage est difficilement interprétable, et l'importance relative des coefficient obtenus est assez floue.

Néanmoins, ce codage est indispensable en analyse d'images, pour faciliter la plupart des actions globales (filtrage de bruit, traitement optique...). Il est tout-à-fait inadapté au dessin ou aux images programmées.



2-5-3 Codage cellulaire ([Jor 74], [Bar 74])

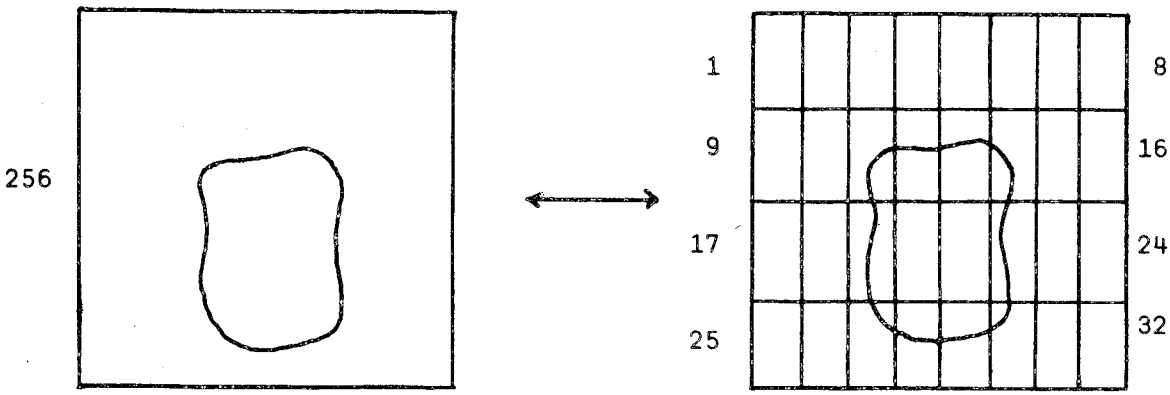
Nous abordons ici des méthodes géométriques de codage, qui sont de manière générale peu adaptées aux images photographiques, mais répondent par contre souvent bien aux exigences des dessins et graphiques.

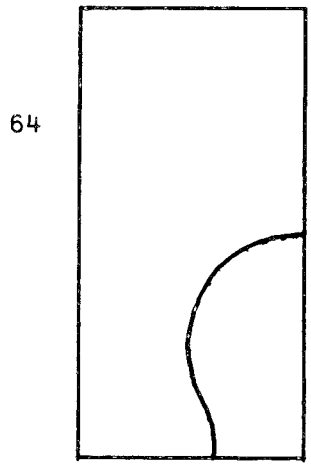
Le codage cellulaire consiste :

- à découper le rectangle d'affichage (ou l'espace de travail) en cellules élémentaires beaucoup plus petites

- à coder chaque cellule ainsi définie, qui comporte donc peu de points, suivant un procédé approprié. Le code résultant est donc une suite de codes des cellules composant l'image.

Ainsi si l'image initiale comporte 256 lignes de 256 points, on peut la découper en 8 x 4 cellules de dimension 32 x 64 ; chaque cellule peut être codée par transitions (voir § 257) sur un code à 5 bits.





une cellule élémentaire

(ici la cellule 11)

Nous obtiendrons le code suivant :

1 à 10 : vide
11 :
12 :
13 :
14 :
15 à 18 :
etc...

} description en transitions

Il est clair sur cet exemple que la taille du code généré dépend :

- de la complexité de l'image
- de la taille de la cellule
- du procédé de codage de la cellule élémentaire

En ce qui concerne les taux de compression T_1 et T_2, ils sont difficiles à évaluer, mais nous pouvons dire que

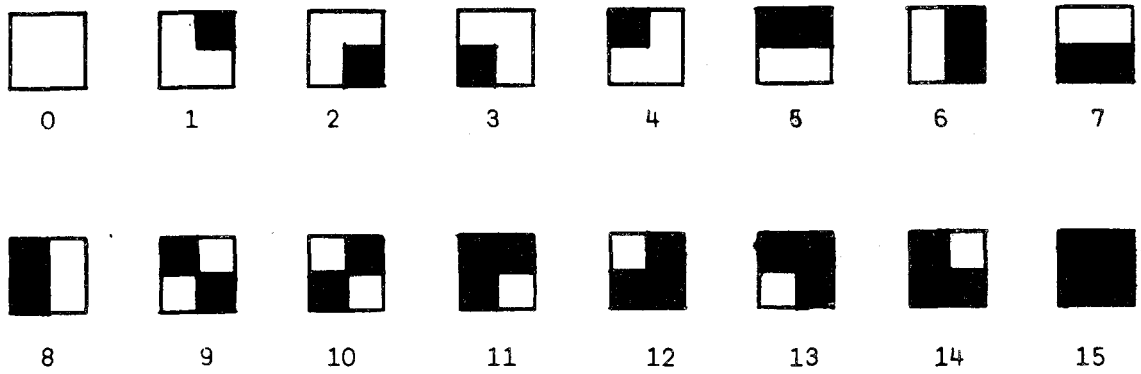
1) si la complexité est faible, le code sera faible
2) si elle est élevée, il pourrait se faire que $T_1 \approx 1$

En outre, la mémoire associée devrait avoir une structure particulière (éléments de longueurs variables) ; toute action globale est impossible et les actions locales sont difficiles car elles concernent en général plusieurs cellules.

Ce procédé semble donc peu intéressant, car il est complexe et assez lourd à mettre en oeuvre ; mais une version particulière présente un intérêt certain.

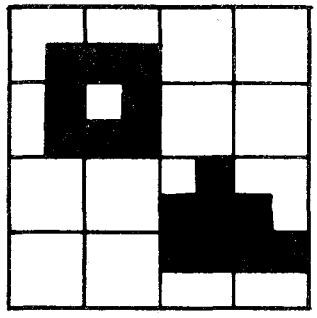
2-5-4 Générateur de caractère étendu

Il s'agit en effet d'un codage cellulaire, où chaque cellule est repérée par un numéro de configuration. Ainsi, pour une cellule 2 x 2, on a 16 configurations possibles :



Le code de configuration occuperait 4 bits.

Exemple



description

2	7	0	0
6	12	0	0
0	0	12	3
0	0	5	5

d'où I = (2, 7, 0, 0, 6, 12, 0, 0, 0, 0, 12, 3, 0, 0, 5, 5)

pour une cellule de base n x n, chaque case peut avoir deux configurations (0 ou 1) dans le cas d'une image binaire. Il y a donc 2^{n^2} configurations théoriques possibles, codées sur n^2 bits

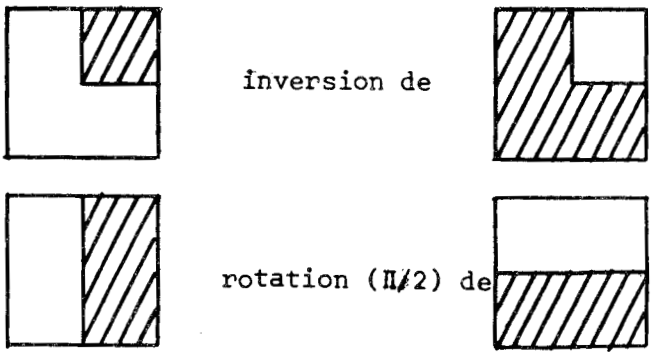
n	2	3	4	8	16
nombre de configurations	16	512	64 K	$\neq 16 \times 10^{18}$	$\neq 64 \times 10^{75}$

Il n'est donc pas raisonnable de dépasser une taille élémentaire de quelques unités.

Il est clair que si nous ne limitons pas le nombre de configurations, la description totale contiendra autant de bits que la description matricielle. La méthode brute n'aurait donc aucun avantage, on cherche alors à réduire le nombre de configurations en tenant compte de :

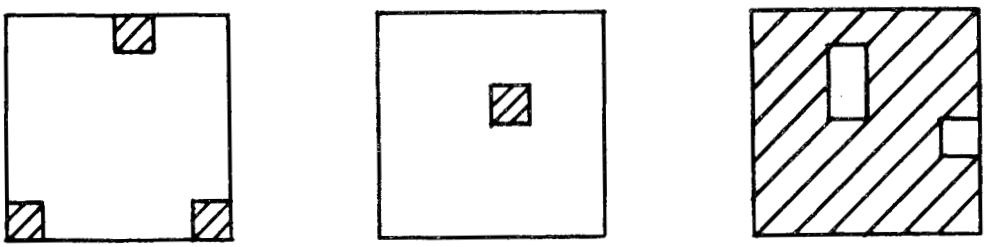
- un certain nombre d'opérations permettant de déduire une configuration d'une autre, à savoir rotation, symétrie, translation, inversion, union...

exemples :



- une contrainte aisément applicable dans le cas d'une image graphique, consistant à imposer une continuité dans les motifs décrits

exemple de configurations "interdites" :



La réduction d'un facteur 2 du nombre de configurations permet de gagner 1 bit sur le code de chaque cellule (et 1 bit seulement). On peut aussi limiter le vocabulaire ainsi autorisé, et le codage des données alphanumériques est un bon exemple ; il faut néanmoins toujours disposer de la correspondance

n° de configuration<—>configuration

c'est-à-dire d'une mémoire morte importante.

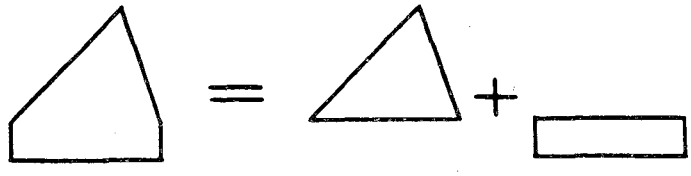
2-5-5 Codage par grammaire de polygones ([Dac 71])

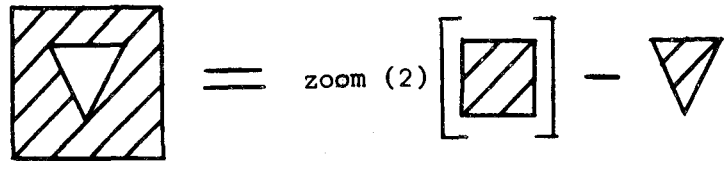
Ce procédé n'est utilisé qu'en analyse d'image et en reconnaissance de formes. Il est basé sur l'existence d'un langage graphique composé :

1) d'un alphabet, constitué de symboles graphiques élémentaires

exemple :

2) de règle de construction (syntaxe), comme par exemple : union intersection, rotation, changement d'échelle
exemples :





Ce procédé est limité en objets constructibles par l'alphabet de base. Il est lourd d'emploi et ne se justifie que dans des applications spécifiques. Les critères que nous avons définis sont malaisés à évaluer. Plutôt que de codage, il est ici préférable de parler de langages graphiques.

2-5-6 Approximations analytiques ([Ger 76],[And 76],[Jor 73])

L'image doit ici se composer de courbes ou de surfaces (courbes ou planes). Le codage consiste alors à trouver des coefficients numériques caractérisant complètement chaque courbe ou surface, partiellement ou en totalité. Ainsi pouvons-nous chercher à approcher des (portions de) courbes à l'aide des fonctions polynomiales habituelles, dont nous calculons au mieux les coefficients. Si a_i et b_i sont des coefficients de la description paramètrique d'une courbe c, nous aurions

$$x(t) = \sum_{i=0}^{n} a_i \, t^i$$

$$y(t) = \sum_{j=0}^{m} b_j \, t^j$$

Deux remarques s'imposent :

1) la précision dépend de l'ordre des polynômes et donc du nombre de coefficients
2) en cas de forme très complexe, soit l'ordre devient très élevé, soit on découpe en portions plus petites

Des méthodes beaucoup plus efficaces ont été mises au point ; citons notamment les courbes de Bézier et les approximants Splines. Elles ont des applications particulières, comme par exemple l'étude de formes d'avion ou de carrosseries automobiles, mais ne peuvent être considérées comme générales ; elles demandent des calculs souvent longs, difficilement réalisables localement.

2-5-7_ Codage par transitions ([Hua 75])

Analysons ligne par ligne une image matricielle : chacune est décrite par une suite de couples (abscisse, couleur) des différentes variations de couleur.

exemple : _

0	0	0	1	1	0	0	0
0	0	1	1	1	0	0	0
0	0	1	3	3	1	0	0
0	0	1	3	3	3	1	0
0	0	1	1	1	1	1	0
0	0	1	1	1	0	0	0
0	0	0	1	1	0	0	0
0	0	0	0	0	0	0	0

⇔
(0), 3(1), (0)
(0), 2(1), 5(0)
(0), 1(1), 3(3), 5(1), 6(0)
(0), 2(1), 3(3), 6(1), 7(0)
(0), 2(1), 6(0)
(0), 2(1), 5(0)
(0), 3(1), 5(0)
(0)

chaque abscisse est codée sur 3 bits
chaque valeur est codée sur 2 bits

taille du code = 26 segments x (3 + 2) = 130 bits

Pour une ligne de longueur $n = 2^k$, avec un code de couleur sur q bits, chaque couple occupe k + q bits ; le nombre total de transitions est égal à la complexité p de l'image, aux portions horizontales près. Donc la taille du code est

$$c = p \times k \times q$$

Application numérique : $k = 8$, $q = 8$ $\Rightarrow c = 16\ p$

D'où le taux de compression : $T_2 = c/p = 16$, ce qui est élevé. Néanmoins, dans le cas d'une image peu complexe, ou dans celui d'un terminal à balayage télévision (où justement l'analyse est faite ligne par ligne), ce système peut être interessant ; il est tout-à-fait inadapté aux images photographiques, où les variations de couleur sont nombreuses et aléatoires.

Ce type de codage inclut aussi le "Run Length Encoding", ou 'codage par longueurs' où le code est une suite de couples (longueur, couleur) ; en général, on considère l'image comme séquentielle, et une analyse préalable de la longueur moyenne des segments permet un choix optinal de la longueur en bits du code de longueur. Dans l'exemple précédent, nous aurions 19 segments, dont les longueurs seraient :

3, 2, 5, 3, 5, 1, 2, 1, 4, 1, 3, 1, 3, 4, 4, 3, 6, 2, 11

Soit une longueur moyenne de 44/19 ≠ 3,3

codons sur 2 bits, suivant

code	0	1	2	3
longueur	1	2	3	4

Nous obtenons le codage suivant

longueur	2	1	30	2	30	0	1	0	3	0	2	0	2	3	3	2	31	1	332
couleur	0	1	0	1	0	1	3	1	0	1	3	1	0	1	0	1	0	1	0

soit un total de { 24 codes à 2 bits = 48
19 couleurs à 2 bits = 38
86 bits

Si nous avions choisi de coder sur 3 bits, nous aurions eu de même :

20 codes à 3 bits : 60
19 couleurs à 2 bits : 38
98 bits

Remarquons qu'un codage matriciel demanderait

8 x 8 x 2 = 128 bits

Mais ce codage "astucieux" a le gros inconvénient de rendre absolument impossible toute action sur l'image et de provoquer des catastrophes en cas d'erreur, car tout le codage est relatif.

2-5-8 Codage prédictif ([Who 61])

Ce terme recouvre des méthodes particulières souvent utilisées en transmission numérique d'images. Il s'agit

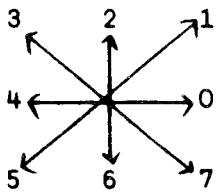
- soit d'intra-prédiction : on prédit une ligne par la connaissance de la (ou les n) ligne(s) précédente(s)
- soit d'inter-prédiction : même méthode d'image(s) à image.

Ne sont transmises que les différences par rapport à l'image prédite. Mais ce procédé "par dérivée", analogue à celui décrit en 257, à les mêmes inconvénients : sensibilité aux erreurs, difficulté d'interprétation. Il est aussi difficile à mettre en oeuvre, et on le réserve généralement a des applications particulières (compression de bande en télévision numérique par exemple)

2-5-9 Codage par contours ([Fre 61])

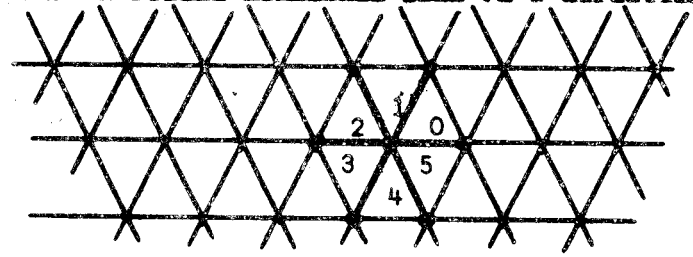
Les contours des objets de l'image étant supposés définis, on choisit pour un contour donné un point de départ (x_o, y_o). Puis on passe d'un point au point suivant du contour à l'aide d'un code de direction, jusqu'au retour en (x_o, y_o). Ainsi, les directions pourraient être représentées par :

- dans un réseau rectangulaire plan (à 8 directions)



(codage à 3 bits)

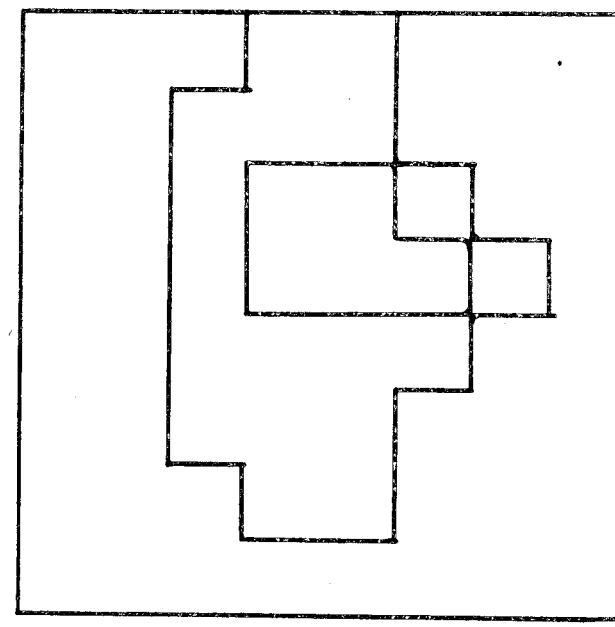
- dans un réseau hexagonal plan (à 6 directions)



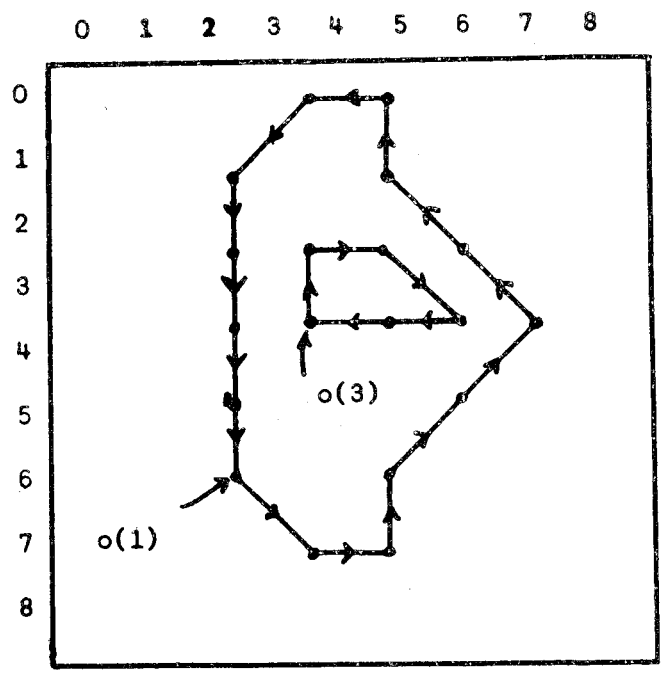
Reprenons l'exemple de § 257, dans un réseau à 8 directions.

0	0	0	1	1	0	0	0
0	0	1	1	1	0	0	0
0	0	1	3	3	1	0	0
0	0	1	3	3	3	1	0
0	0	1	1	1	1	0	0
0	0	1	1	1	0	0	0
0	0	0	1	1	0	0	0
0	0	0	0	0	0	0	0

image source



contours



Couleur	code	taille
0	/	2
1	(5,2)	2+6
	7,0,2,1,1,	
	3,3,2,4	
	5,6,6,6,6,	14 x 3
3	(3,3)	2+6
	2,0,7,4,4	5x3
	total	75 bits

suivi des contours code correspondant

Nous pouvons d'ailleurs imaginer des codages dans l'espace à trois (voire quatre !) dimensions ; ils sont en tout cas caractérisés par

- l'espace utilisé (dimension et maillage)
- le nombre de directions autorisées
- l'existence de facteurs de répétition
- l'existence d'opérateurs élémentaires (mise à l'échelle, rotation, inversion, etc...)
- l'utilisation de directions absolue et/ou relative

Ce codage est tout-à-fait adapté et appliqué effectivement aux images de nature graphique ; il permet de structurer facilement des données graphiques, de les manipuler. Il est particulièrement adapté à certaines opérations (translation, rotations particulières...) et permet des calculs sur les objets (périmètre, polygone convexe englobant, surface,etc...). La quantité de code est comparable à la complexité de l'image.

Toujours dans le cas d'une image graphique, le processus de codage est presque immédiat, quant au décodage, il est assez simple, car il consiste à calculer (x_i, y_i) à partir de (x_{i-1}, y_{i-1}) et du code. Ce procédé est particulièrement adapté aux terminaux fonctionnant en mode vecteur, et, au prix de quelques aménagements que nous détaillerons, aux terminaux à balayage télévision.

2-5-10 Comparaison ([Hub 75])

Il convient maintenant de réunir dans un tableau récapitulatif les données se rapportant aux critères d'évaluation que nous avions défini, à savoir

- l'adéquations aux images, photographiques ou graphiques
- l'adéquations aux actions, globales ou locales, que l'on voudrait réaliser
- les taux de compression

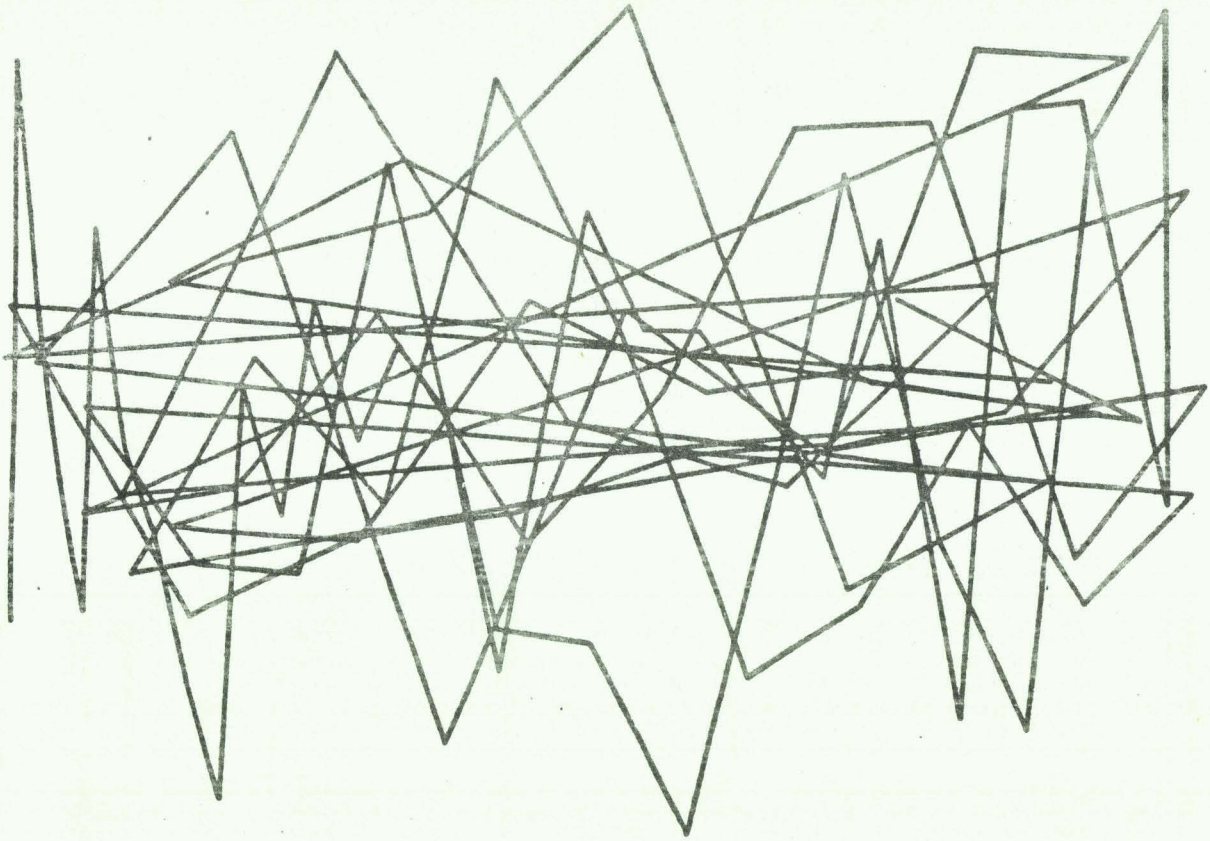
$$T_1 = \frac{q\ (\text{matrice})}{q\ (\text{code})}$$

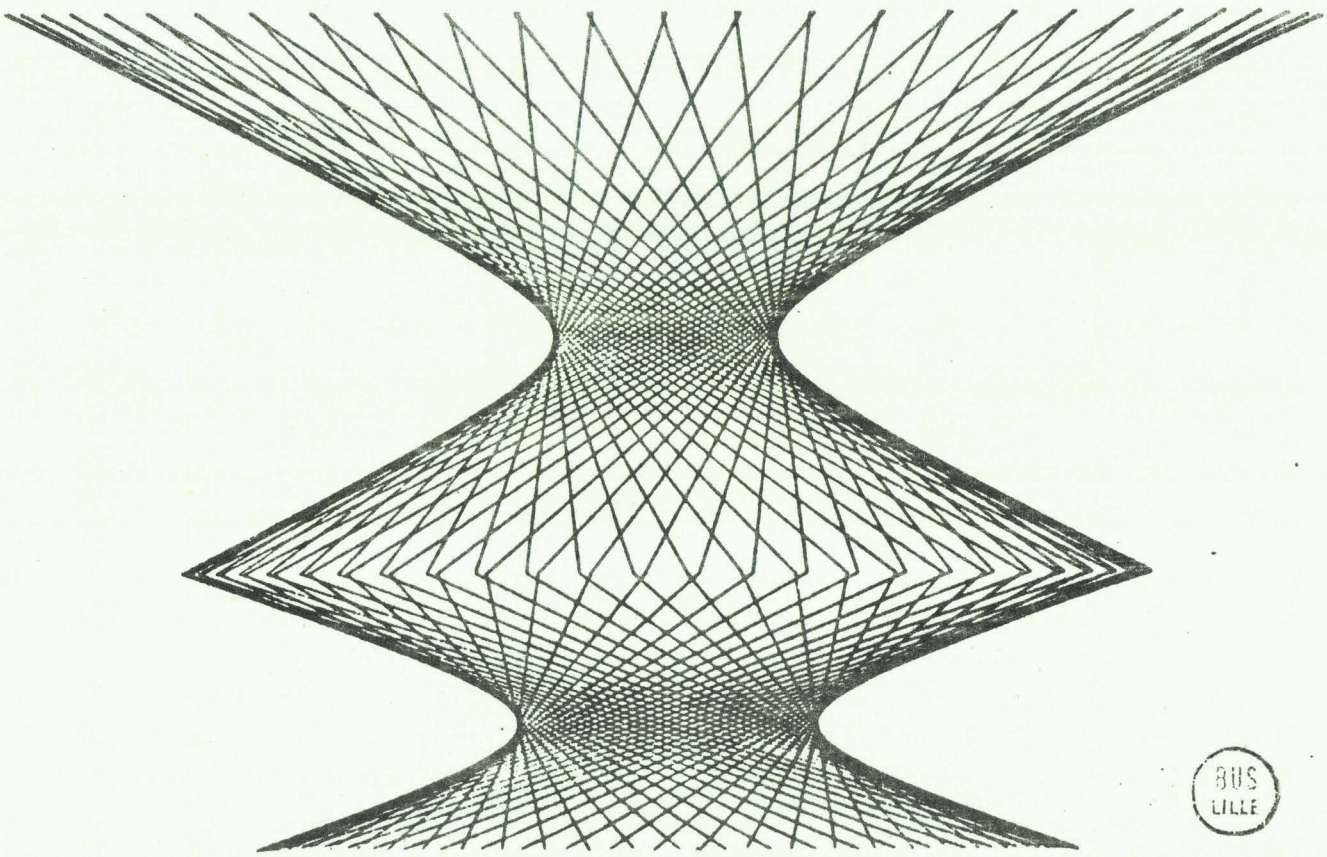
$$T_2 = \frac{q\ (\text{code})}{\text{complexité}}$$

- la difficulté du codage
- la difficulté du décodage et la mémoire associée (dans le cas d'un écran à rafraîchissement)

Codage	Images		Actions		Compression	
	Photographique	Graphique	Globales	Locales	T_1	T_2
MATRICE	Bon	Moyen	Oui	Non	1	>> 1
Changement d'espace	Bon	Mauvais	Oui	Non	?	?
Cellulaire	Mauvais	Moyen	Non	Difficile	à peu près constant	
Générateur de caractère	Mauvais	Bon	Non	Difficile	constant	
Grammaire	Mauvais	Moyen	Difficile	Oui	>> 1	$\underline{\sim}$ 1(?)
Approximations analytiques	Mauvais	Bon	Difficile	Oui		
Transitions	Moyen	Bon	Non	Non	> 1	$\underline{\sim}$ 1
Prédictif	Bon	Moyen	Oui	Non	>> 1	
Contours	Mauvais	Bon	Non	Oui	> 1	$\underline{\sim}$ 1

Processus de Codage	Processus de Décodage	Mémoire associée	Codage
/	/	taille élevée vitesse élevée	MATRICE
difficile	difficile	taille élevée	Changement d'espace
difficile	facile	structure particulière	Cellulaire
difficile	facile	taille faible, mais mémoire morte importante	Générateur de caractère
difficile	difficile		Grammaire
facile	facile	taille assez élevée, vitesse élevée	Approximations analytiques
difficile	assez facile	taille très élevée	Transitions
assez facile	nécessité de processeurs rapides	taille assez faible	Prédictif





CHAPITRE 3

-:-:-:-:-:-:-:-:-:-:-:-:-

LE SYSTEME PROPOSE

3 LE SYSTEME PROPOSE

3-1 Objectif.

Plusieurs considérations préliminaires ont présidé à l'étude de ce système. Elles concernent à la fois le type d'images à visualiser, les actions susceptibles d'être réalisées, et aussi certaines caractéristiques du terminal.

3-1-1 Images à visualiser

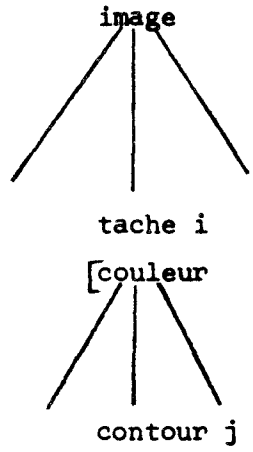
L'hypothèse faite est la suivante : l'image apparaît sur l'écran comme un ensemble de taches de couleur

- formant une partition de l'écran
- caractérisées par 1) leur contour
  2) une couleur ou un algorithme simple de calcul de la couleur à l'intérieur de la tache.

Une question se pose alors : quelles images a-t-on ainsi exclues a priori ? Il est clair que cette vision des choses ne concerne pas les images de type photographique dans lès premières étapes de leur traitement, c'est-à-dire lorsqu'elles sont encore floues, bruitées. Mais nous n'excluons en aucune façon de telles images rendues nettes par filtrage numérique, application de seuils ou tout autre procédé, pourvu que l'on puisse définir des contours.

Nous n'excluons pas non plus les tracés de type linéaire, que nous considérons comme des taches de largeur unitaire. De toute évidence nous incluons ainsi des images relevant de la cartographie, de l'électronique (schémas), de l'animation, ou de toute représentation d'ordre topographique.

Sans préjuger de la structure logique de l'image que l'utilisateur pourrait y superposer, la structure physique sur l'écran est la suivante :



Remarque : une tache peut en effet se décomposer en plusieurs contours, par exemple si elle comporte un trou, ou si elle est manifestement constituée de plusieurs éléments disjoints.

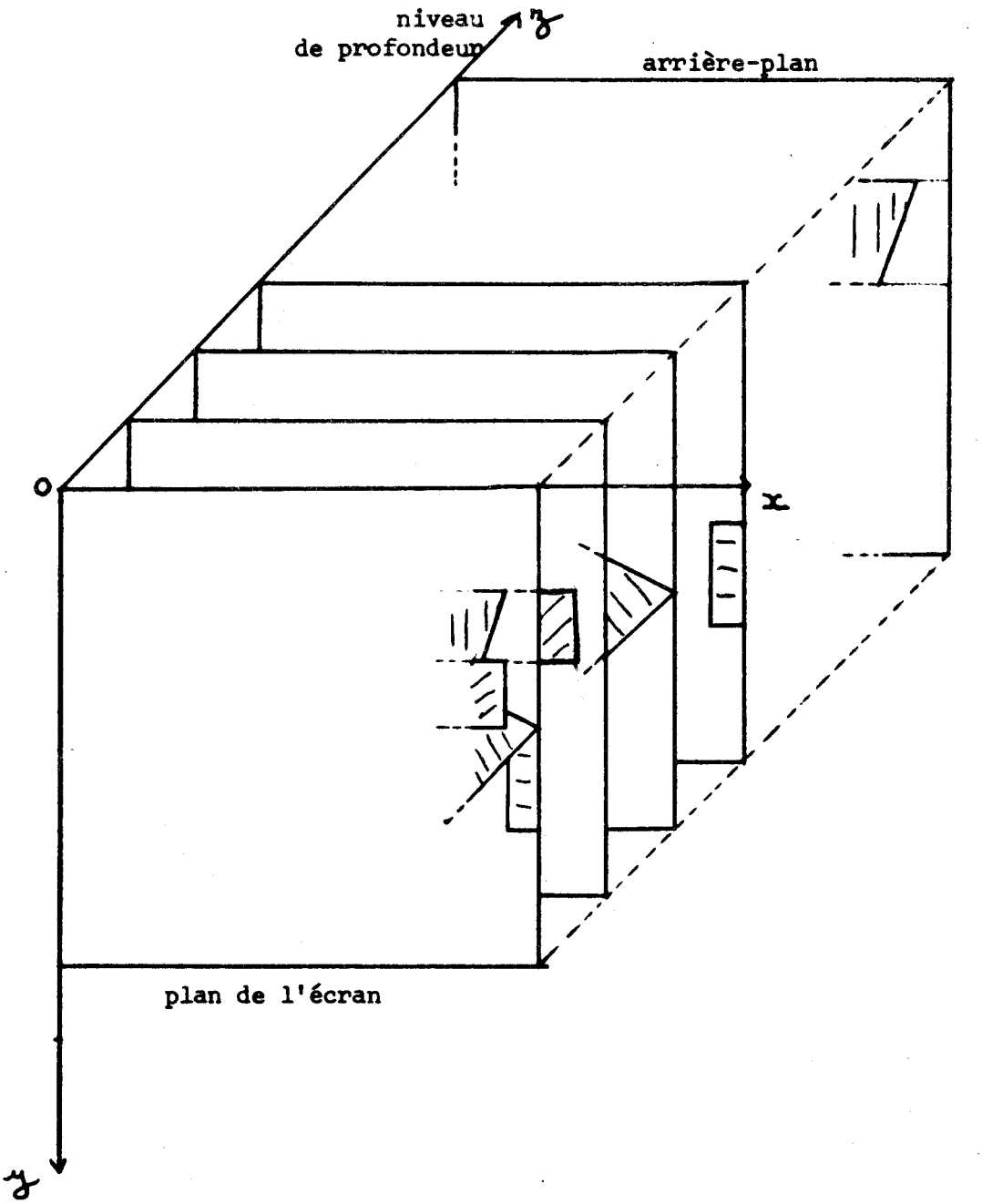
3-1-2 Actions à réaliser

Nous n'imposons aucune limite à ces actions, mais il paraît évident d'affirmer que l'hypothèse ci-dessus privilégie les actions locales, c'est-à-dire consistant à créer, désigner, manipuler, effacer une ou plusieurs taches de l'image, et nous éloigne des actions globales familières à l'analyse d'images.

Le système doit pouvoir d'autre part supporter une forte interactivité. Si tel est le cas, il se pose un problème important : supposons que nous voulions ajouter une tache en un endroit où une autre préexiste ; quelle sera la tache visible ? Nous voyons ainsi apparaître une notion de profondeur, ou de transparence, ce qui revient au même, proche du problème des lignes et surfaces cachées. Si nous restons dans l'hypothèse de taches planes, il convient alors d'associer à chaque tache un repère de profondeur, que nous appelerons "niveau", et qui permettra cette superposition de taches. On peut alors parler de représentation volumique, ou tri-dimensionnelle ; si la réalisation matérielle assure selon un certain algorithme la projection des taches superposées, il suffit que

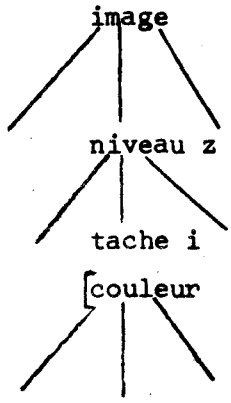
- cet algorithme soit connu de l'usager
- il soit assez souple pour permettre diverses manipulations sur les taches concernées.

L'espace de travail est donc un ensemble de plans, parallèles à l'écran, et l'image finalement affichée sera le résultat obtenu après projections successives de ces plans sur le plan de l'écran.

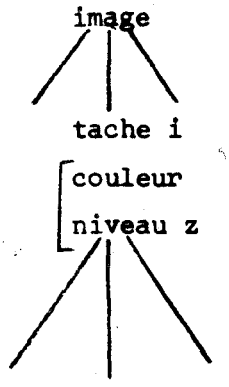


Il serait également souhaitable de l'effacement d'une tache intermédiaire restitue l'arrière-plan qu'elle cachait.

Dans l'espace de travail, la structure de l'image devient :



ou encore, ce qui peut paraître plus intéressant :



3-1-3 Le système

Il est souhaitable de demander à la partie locale du système les caractéristiques suivantes :

- forte autonomie, et donc existence d'un processeur local évolué
- "intelligence" élevée et temps de réponse faible
- possibilité d'adjonction de tous les périphériques et moyens de communication souhaitables.
- matériel peutencombrant
- universalité de la liaison vers le processeur éloigné.

3-2 Description du système.

3-2-1 L'écran ([Lim 77], [Kre 73], [Mat 78])

3-2-1-1 Choix

Nous cherchons un écran réunissant les qualités suivantes :

- affichage en couleurs, assez nombreuses. En effet, la facilité d'utilisation et d'interprétation de l'information couleur n'est plus à démontrer : cartographie, schémas électriques, plans divers en sont la preuve. Pour une utilisation usuelle, trente à cinquante couleurs différentes paraissent suffisantes ; pour des applications concernant la perception visuelle, il semble qu'un millier de teintes soit un ordre de grandeur déjà satisfaisant
- temps de réponse propre faible, dans le cas où l'interactivité est importante. Ceci exclut les dispositifs à mémoire actuels, dont le temps de réponse est prohibitif dès que l'image est un tant soit peu complexe
- faible coût et transportabilité

Nous nous orientons donc naturellement vers le dispositif d'affichage le plus commun qu'il soit, à savoir l'écran de télévision, qui répond bien aux contraintes ci-dessus :

- le nombre de couleurs peut être élevé (ex. : 4096 dans le système COLORIX), avec une qualité assez satisfaisante
- le temps de réponse propre est égal à la période de rafraîchissement de l'écran (20 ms ; cf. § 3-2-1-2), ce qui est excellent, (5 fois plus court que la rémanence visuelle)
- la transportabilité est évidente. Quant au coût, c'est le plus bas par rapport aux moniteurs professionnels ou aux tubes à mémoire

De plus :

- les téléviseurs couleurs étant identiques (à une norme de codage près, SECAM, PAL ou NTSC), on peut les interchanger, ou les multiplier (pour l'enseignement, par exemple)
- les techniques habituelles des terminaux alphanumériques peuvent être facilement appliquées

En contrepartie, il faut noter :

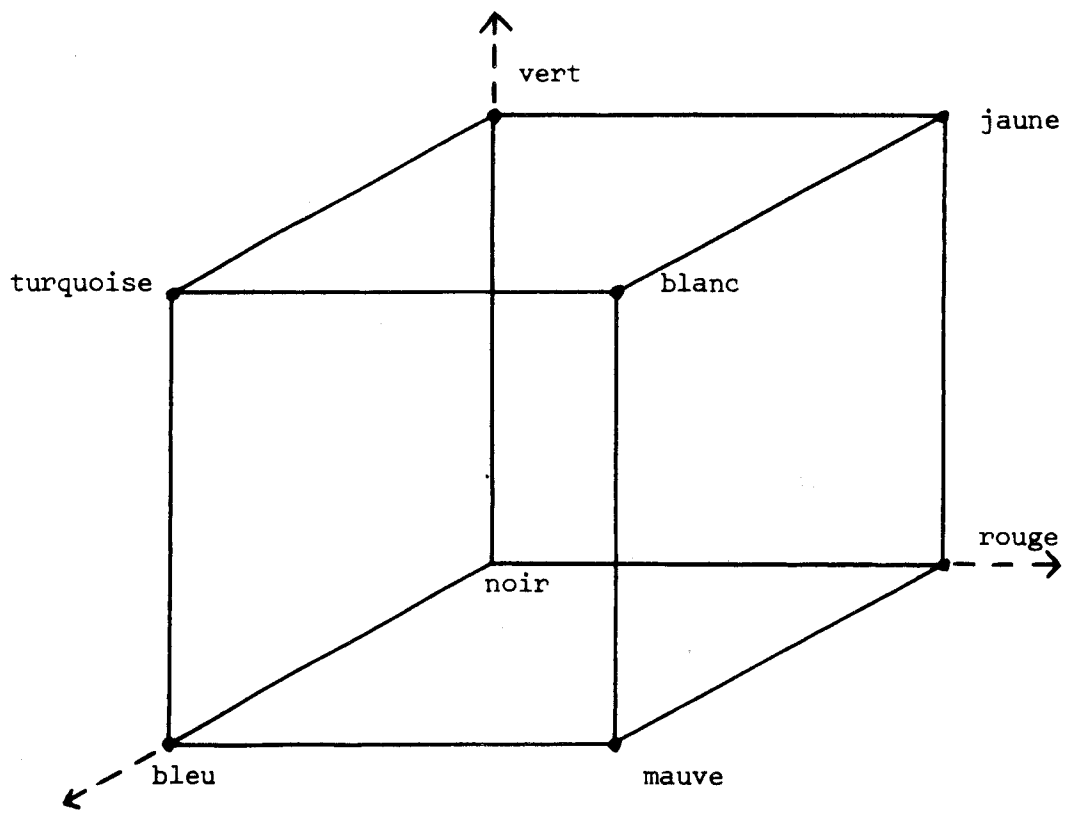
- une résolution assez faible : la limite théorique est de 625 lignes/ En fait l'affichage de 512 lignes de 512 points est déjà illusoire (cf. § 3-2-1-2)
- un scintillement, qui est assez désagréable et fatiguant si l'on observe longtemps l'écran, d'assez près
- la nécessité de réaliser un rafraîchissement, c'est-à-dire de renvoyer périodiquement (toutes les 20 ou 40 ms) sur l'écran le contenu d'une mémoire contenant l'image

3-2-1-2 Rappel sur le fonctionnement du téléviseur

* Le tube proprement dit :

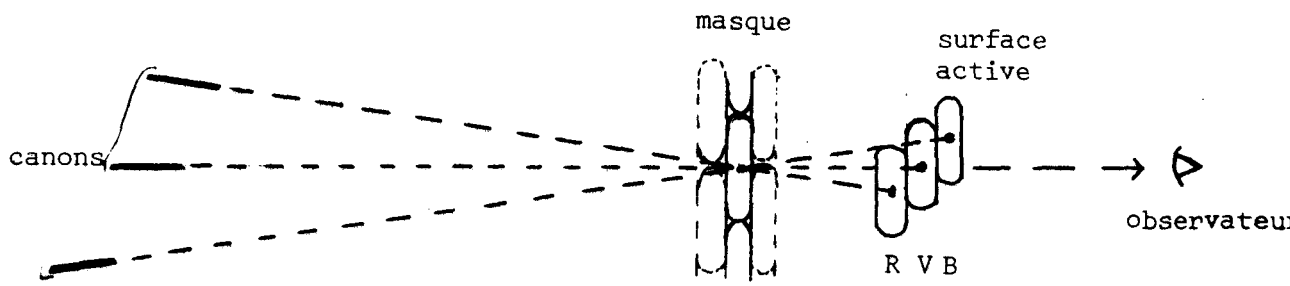
Le tube d'un téléviseur couleur (modèle PIL) est constitué

- d'une surface active supportant les triplets de luminophores rouges, verts et bleus permettant de reconstituer (par synthèse) l'ensemble des couleurs du spectre visible



cube des couleurs

- d'un masque perforé (un trou par triplet de luminophores)
- de trois canons à électrons (un par couleur primaire)



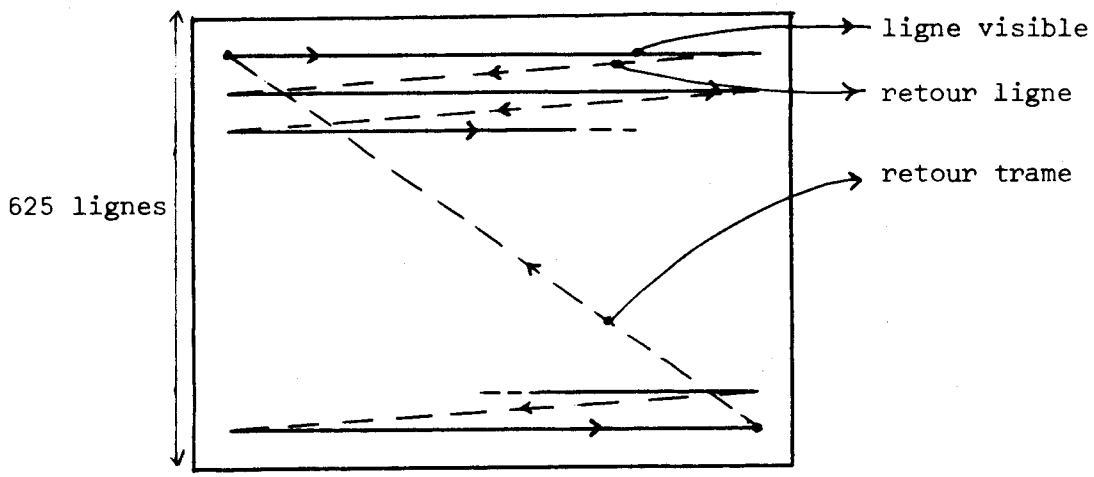
Les commandes appliquées aux trois voies R, V, B, donnent donc une certaine teinte globale pour le triplet correspondant.

Chaque triplet occupe (pour des raisons essentiellement mécaniques) une surface d'environ 1mm x 1mm (sur un tube de taille habituelle).

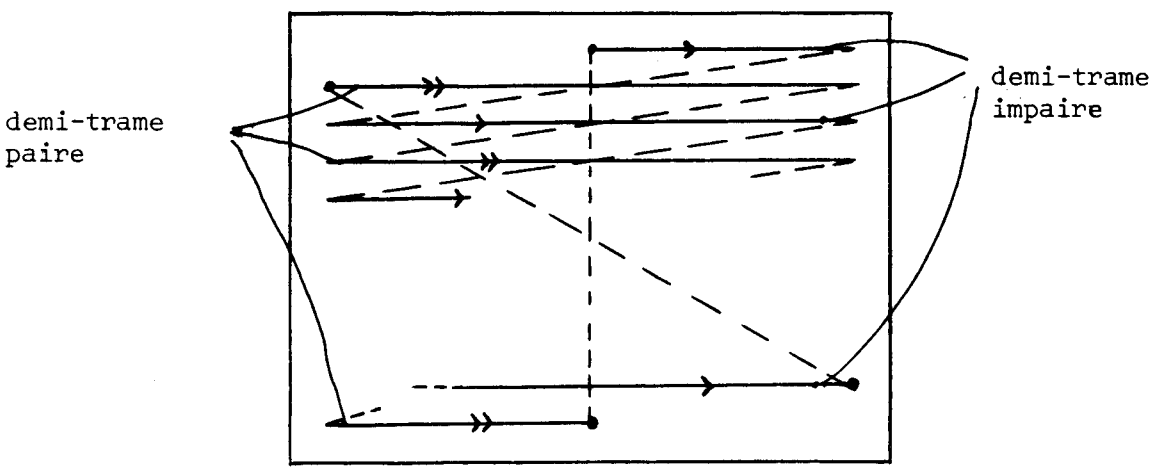
Il est donc illusoire d'espérer visualiser plus de 300 lignes de 400 points (environ). En se limitant à 256 lignes de 256 points pour l'image affichée, un point recouvre à peu près un triplet ; sur un tube de 43 cm de diagonale, cette image occupe environ 28 cm sur 28 cm.

* Le balayage télévision.

L'écran est balayé de haut en bas et de gauche à droite de la façon suivante :



En fait, pour diminuer le scintillement, on balaye par deux demi-trames entrelacées :



Une ligne dure 64 µs (y compris le retour) ; une 1/2trame dure 20 ms. En se limitant à 256 lignes, les deux demi-trames sont identiques.

Pour obtenir une image carrée, on a choisit par commodité :

- durée d'un point : 125 ns
- durée d'une ligne affichée : 256 x 0,125 ns = 32 µs
- temps mort d'une ligne : 32 µs
- durée d'une demi-trame affichée : 256 x 64 µs ≠ 16 ms

D'où le débit demandé au dispositif de rafraîchissement : si on code chaque point sur 6 bits (2 bits par canon soit 64 teintes), on obtient

6 bits/ 125ns : 48 Mégabits/seconde.

* Le codage des couleurs (SECAM).

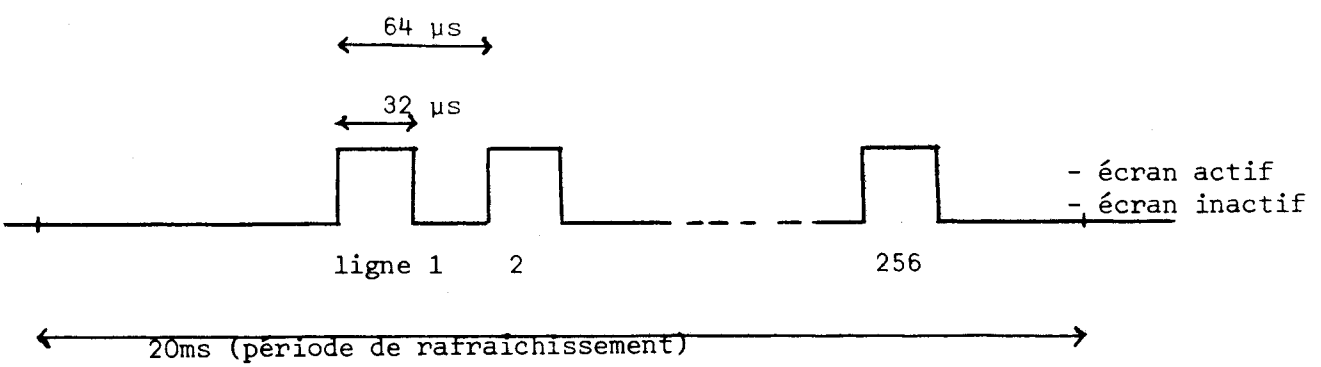
Nous n'allons pas le détailler ici ; retenons simplement que

- ce procédé limite la bande passante en luminance (échelle des gris) et en chrominance (nuances de couleurs) à des valeurs incompatibles avec nos choix (un point/125 ns)
- le codeur est délicat à réaliser

Nous avons donc choisi de transmettre vers l'écran les trois informations séparées (rouge, vert, bleu), à l'aide d'une interface adéquate, assez simple.

3-2-1-3 Implication sur le reste du système

En résumé, l'écran demande donc un rafraîchissement toutes les 20 millisecondes, sous la forme d'un envoi de données (3 fois 2 bits à raison de 2 bits par couleur fondamentale), suivant le diagramme des temps suivant.



3-2-2 Le processeur graphique ([Jor 73], [Ray 76])

3-2-2-1 Choix

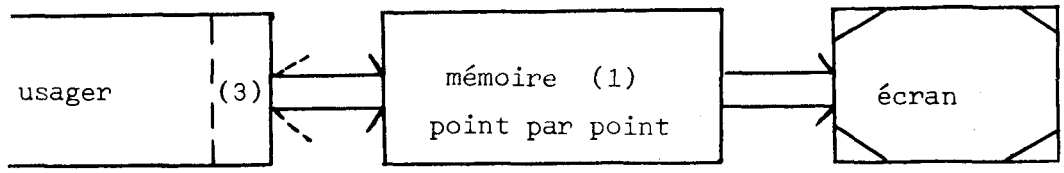
Ce processeur, doit, dans un terminal à rafraîchissement, assurer les taches suivantes :

(1) - mémoriser l'image à afficher

(2) - assurer à l'aide de cette mémoire le rafraîchissement de l'écran

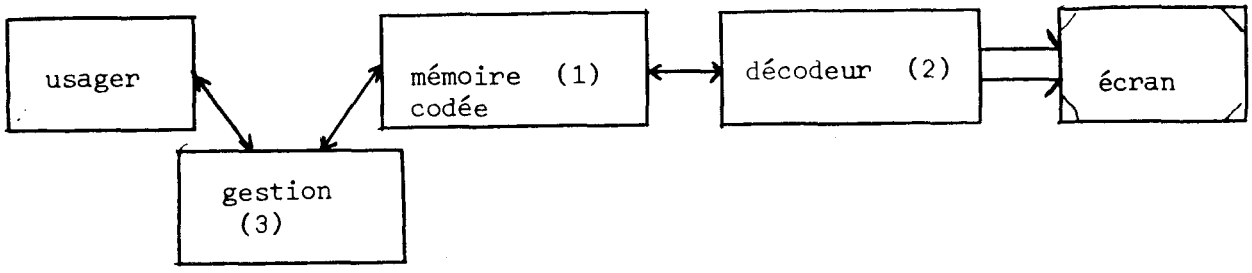
(3) - gérer les accès à cette mémoire

L'architecture d'un système classique serait



(⟹: flux d'information)

Nous avons déjà démontré la nécessité d'un codage, qui nous conduit à l'architecture suivante



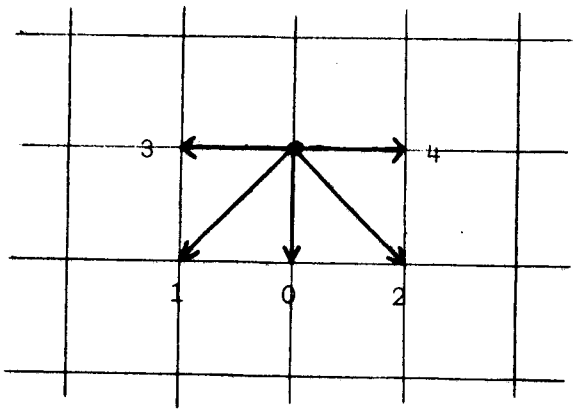
La mémoire est alors de taille plus petite ; les transferts d'informations vus de l'usager sont moindres. Néanmoins, il s'agit de voir si la réalisation de la gestion de la mémoire (3) et du décodeur (1) ne posent pas de difficultés compensant ces avantages.

3-2-2-2 Le décodeur

(a) Codage Il convient tout d'abord de décider d'un procédé de codage satisfaisant

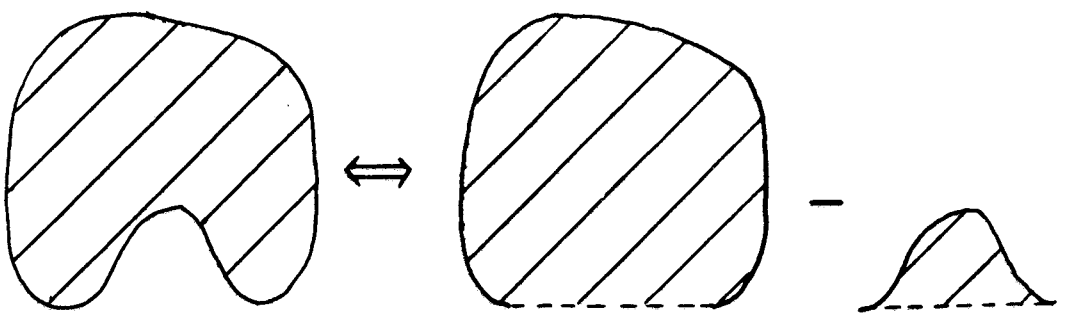
- d'une part l'hypothèse initiale (description par taches)
- d'autre part les contraintes imposées par l'écran

L'hypothèse initiale nous incitait à utiliser un codage par contours (FREEMAN) ; le type de balayage de l'écran nous suggère de l'adapter de la façon suivante : le balayage se faisant toujours de haut en bas, dans une matrice carrée, nous réduisons à 5 le nombre de directions autorisées



Conséquences :

1 : Si un contour de tache "remonte", nous le dissocions en 2 (ou plusieurs) contours :



Nous appelerons "élément" un contour élémentaire

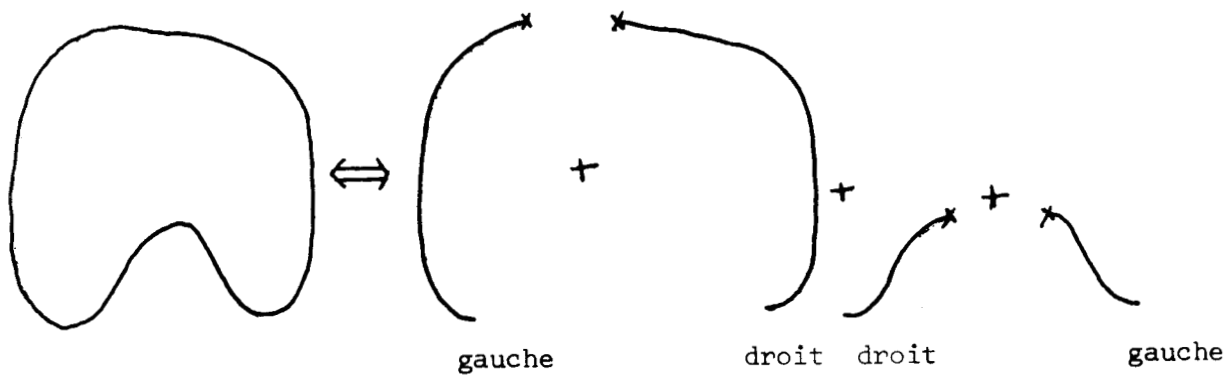
2 : L'origine d'un élément est située en haut de celui-ci.

3 : Il est clair qu'un élément est constitué de deux bords

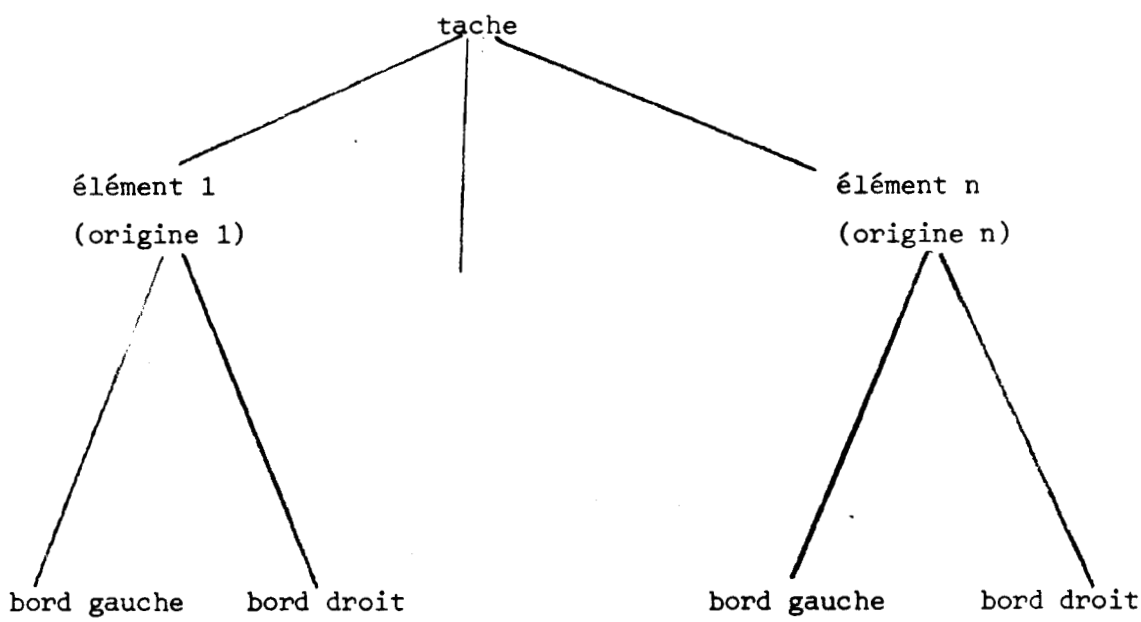
- l'un qui dans la tache initiale est suivi de couleur (si nous suivons le balayage) et que nous appelerons "bord gauche".

- l'autre qui arrête la couleur et que nous appelerons "bord droit"

Ainsi :



4 : En général, l'origine des bords gauche et droit d'un même élément est la même. Nous obtenons donc pour un tache la structure générale suivante



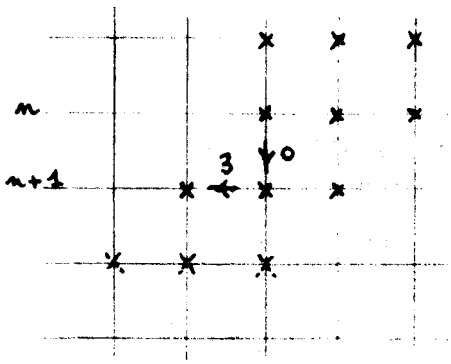
Remarque : Le code utilisé est un code à 5 directions, soit 3 bits par point (si l'on ne tient pas compte des coefficients multiplicateurs). Ces 3 bits autoriseraient 8 directions ; cherchons donc si nous pouvons le ramener à 2 bits, soit 4 directions :

- par raison de symétrie, nous ne pouvons éliminer (1) ou (2) ; nous pouvons éliminer (0) non plus. Restent (3) et (4)

- supposons que nous fusionnons (3) et (4) en une seule direction "h", horizontale, et cherchons un moyen de caractériser le sens (droit ou gauche) du déplacement. Cette direction ne sert que s'il y a un "palier horizontal" sur un bord ; un inventaire de ces "paliers" nous donne les cas suivants :

bord gauche

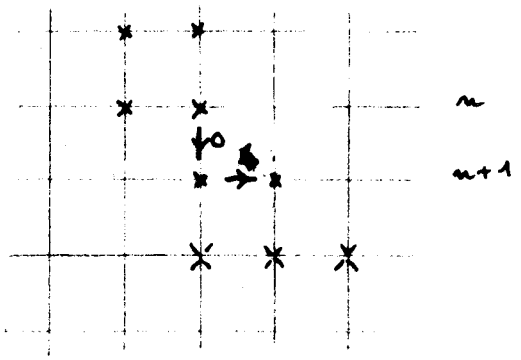
[unité gauche]



(0,3) devient (1)

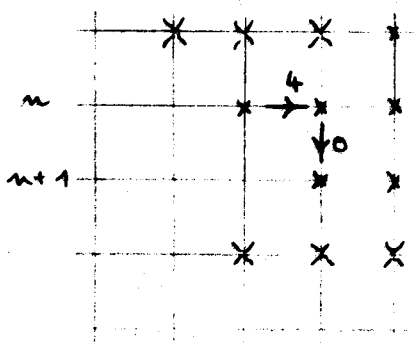
bord droit

[unité droit]



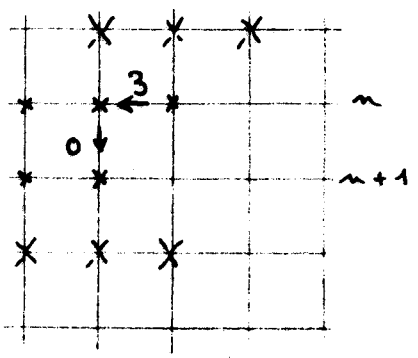
(0,4) devient (3)

[unité droit]



(0,4) devient (2)

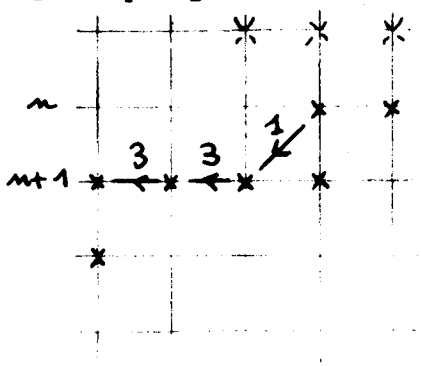
[unité gauche]



(3,0) devient (1)

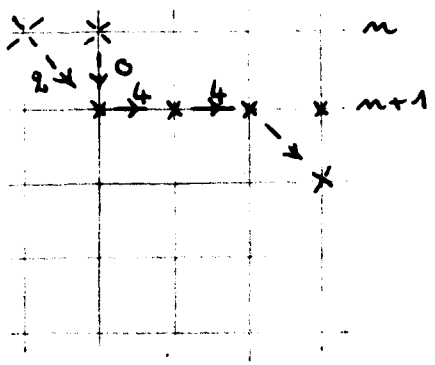
bord gauche

[multiple gauche]



(1) ⟹ déplacement à gauche
code : (1, 3, 3)

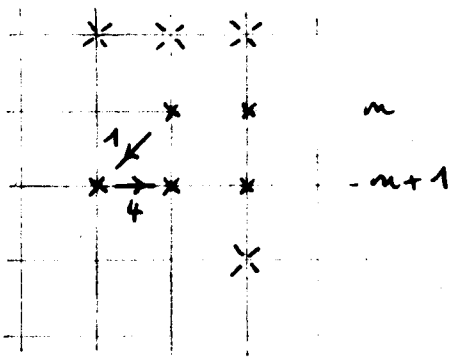
[multiple droit]



(0 ou 2) déplacement à droite

code : (0 ou 2, 3, 3)

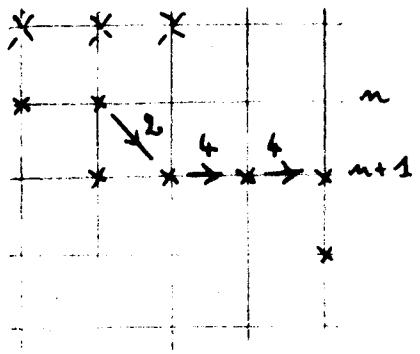
[cas résiduel]



cas interdit

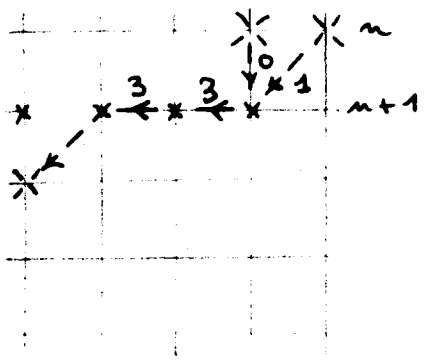
bord droit

[multiple droit]



(2) ⟹ déplacement à droite
cpde : (2, 3, 3)

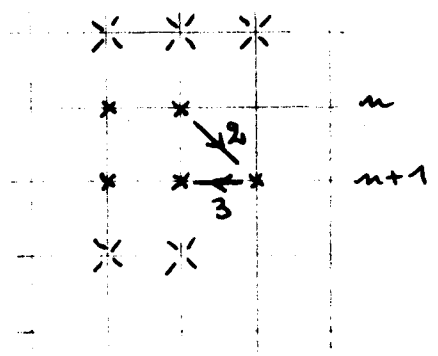
[multiple gauche]



(0 ou 1) déplacement à gauche

code : (0 ou 1, 3, 3)

[cas résiduel]



cas interdit

Nous considérons ces deux derniers cas comme non significatifs (points isolés), et nous réduisons donc le code à 2 bits

0 x inchangé

1 $x \longmapsto x - 1$

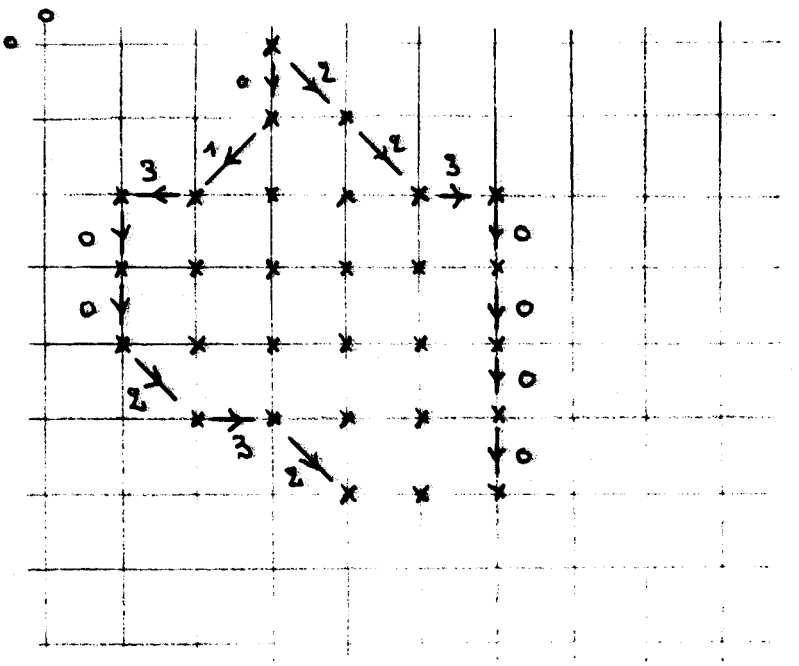
2 $x \longmapsto x + 1$

3 1) bord gauche : [si code initial = 1 : $x \longrightarrow x - 1$
[si code initial ≠ 1 : $x \longrightarrow x + 1$

2) bord droit : [si code initial ≠ 1 : $x \longrightarrow x - 1$
[si code initial = 1 : $x \longrightarrow x + 1$

Le passage d'une ligne à l'autre pourra donc être décrit par un code multiple, dont le premier élément sera toujours 0, 1 ou 2.

Exemple :



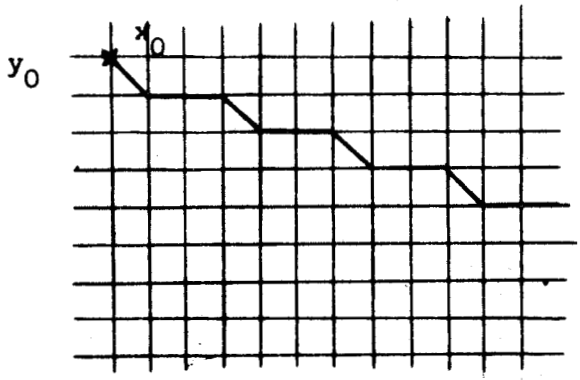
d'où le code :

(3,0) [bord gauche : 0, 1, 3, 0, 0, 2, 3, 2.

origine [bord droit : 2, 2, 3, 0, 0, 0, 0.

Nous pouvons également envisager plusieurs " raffinements"

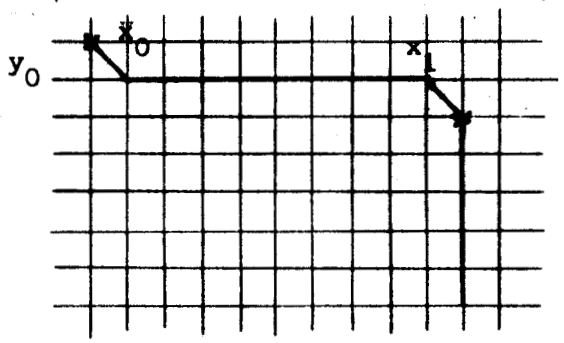
- utilisation de coefficients multiplicateurs de répétition



(x_0, y_0) ; 2, 3, 3, 2, 3, 3, 2, 3, 3, 2, 3, 3.

(x_0, y_0) ; (4) [2, 3, 3]

- génération d'une nouvelle origine en cas de palier très long

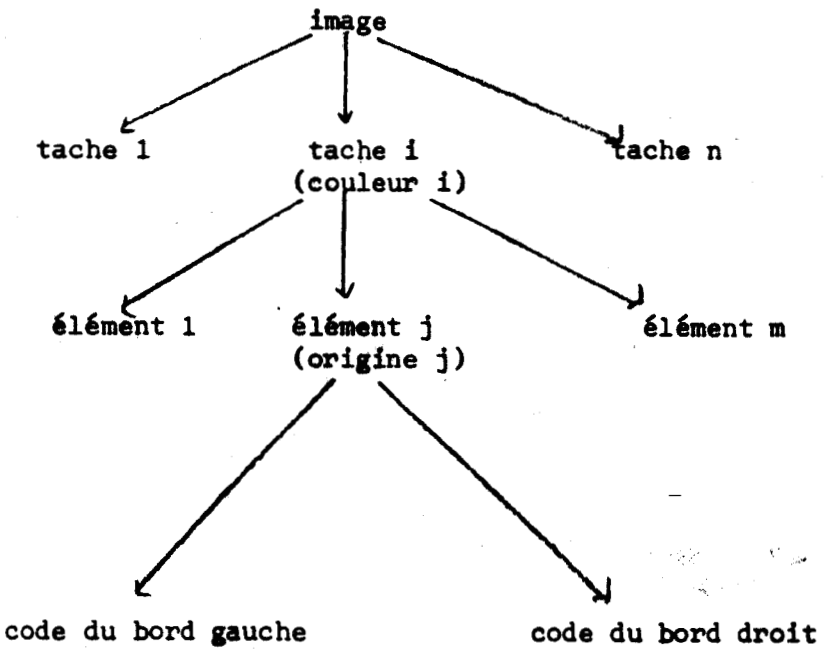


(x_0, y_0), 2, 3, 3, 3, 3, 3, 3, 3, 2, 0, 0, 0..

(x_0, y_0), 2, (x_1), 0, 0, 0...

Mais il est bien évident qu'il faut accepter un compromis entre l'efficacité propre au code, et l'efficacité du décodeur : il n'est pas toujours souhaitable de compliquer le codage, car la structure de donnée associée est plus lourde et moins facile à créer ou à relire.

La structure de donnée est donc, dans la mémoire de rafraîchissement :

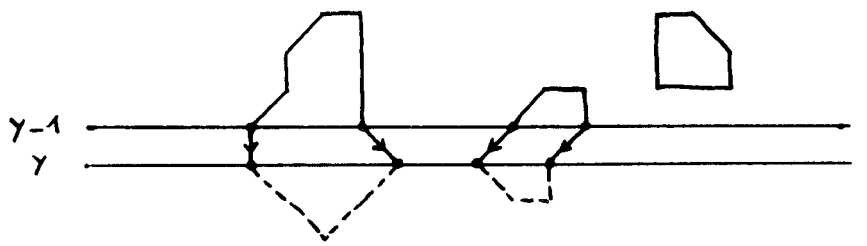


(b) Décodage :

Dans un premier temps, nous ne tenons pas compte des niveaux de profondeur. Le décodeur doit fournir à l'écran, en synchronisme avec celui-ci, la couleur de la tache qui doit être visible au point (x,y). Il fonctionne en y croissants, puis x croissants.

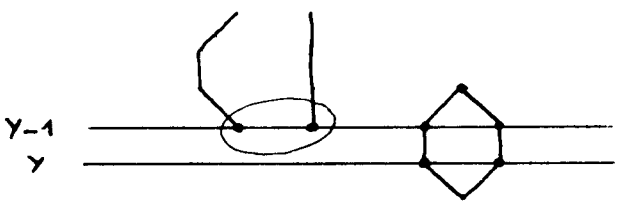
Pour élaborer une ligne "y" donnée, le décodeur doit

- élaborer une liste triée des points courants sur les bords des éléments déjà pris en compte

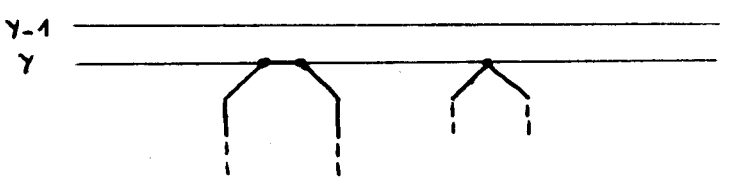


(pour cela il dispose du codage des bords et de cette liste pour la ligne "y-1")

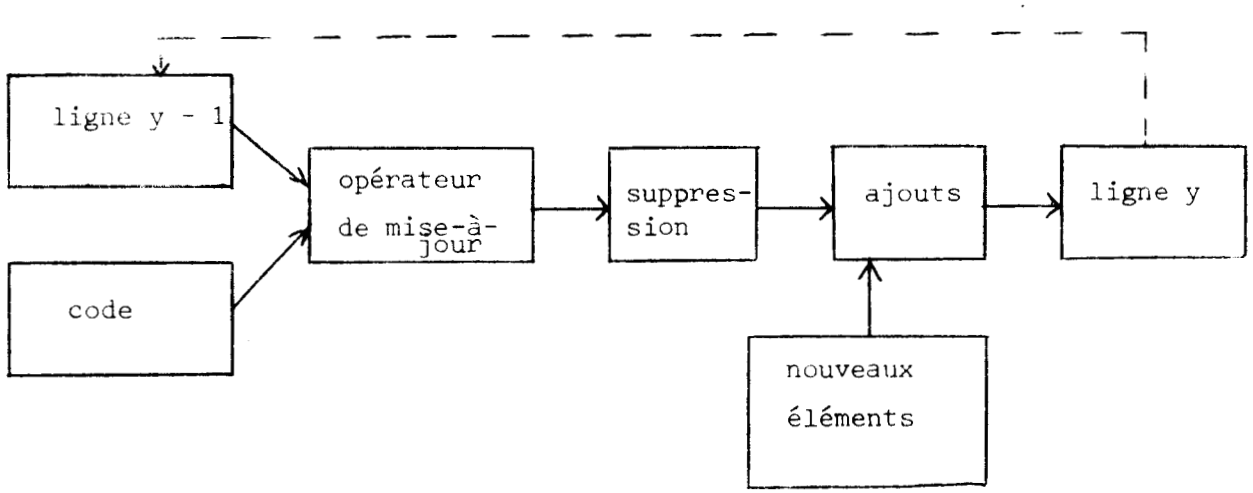
- supprimer de cette liste les bords terminés



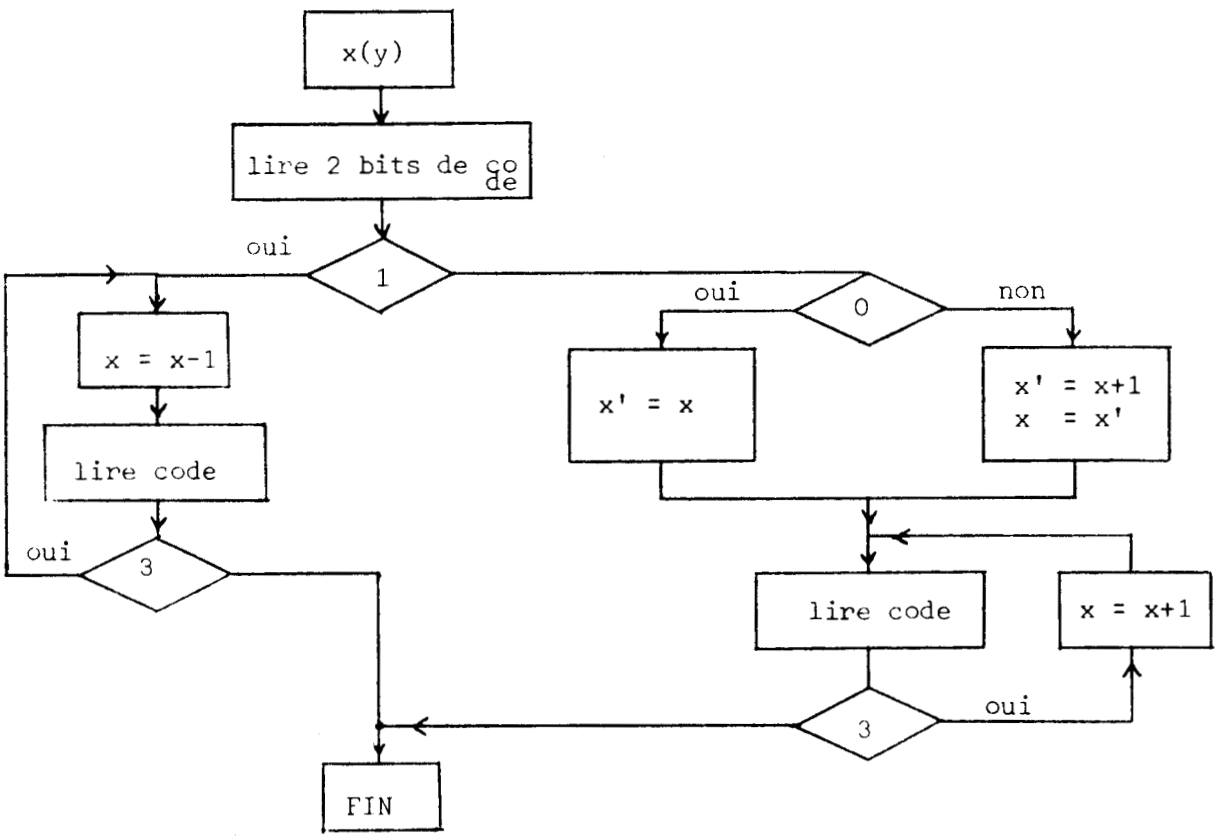
- ajouter les bords des éléments nouveaux à prendre en compte



d'où le schéma fonctionnel suivant



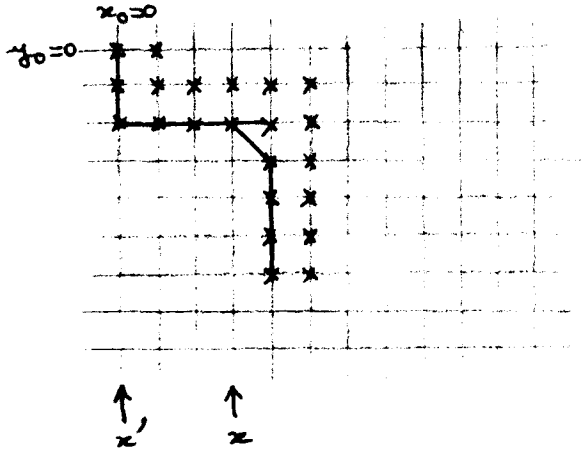
1 . L'opérateur de mise à jour est assez simple : dans le cas où l'on ne tient pas compte des niveaux, il y a partition de l'écran, et les bords ne peuvent se croiser. Il s'agit donc de réaliser pour chaque élément le calcul de x à la ligne (y) à partir de x à la ligne (y-1), en fonction du code. On obtient l'algorithme suivant (ici pour un bord gauche) :



Notes : 1) Si "lire code" est impossible, il y a suppression du contour,

2) . x' est l'abscisse réellement affichée

. x est l'abscisse utilisée pour le calcul à la ligne suivante, qui peut être différente : ainsi par exemple.



(0,0), 0, 0, 3, 3, 3, 2, 0, 0, 0,

ligne	x'	x
0	0	0
1	0	0
2	0	3
3	4	4
4	4	4
5	4	4
6	4	4

2]. L'opérateur d'ajouts dispose des origines des éléments qui démarrent à la ligne y, et les insère dans la liste triée en x des points courants des autres éléments.

3]. La ligne y finale se présente sous la forme d'une succession de données de type (couleur i, abscisse x_i, type de bord), triée en x. Il reste donc à réaliser un "intégrateur", disposant de cette liste en entrée et élaborant la séquence de points finale.

Si nous tenons compte des niveaux, le décodeur doit aussi comparer en chaque point de la ligne les niveaux des éléments concernés pour déterminer lequel est visible.

Dans tous les cas, il est clair que l'on a intérêt à organiser la structure de données

- en ordonnées d'origines croissantes, (et donc par éléments), pour faciliter l'insertion de nouveaux éléments au fur et à mesure que y augmente
- en abscisses d'origine croissante, pour une ligne donnée.

Un rangement supplémentaire par niveaux serait souhaitable également (voir § 4-1)

3-2-2-3 La gestion de la mémoire

Il s'agit de pouvoir

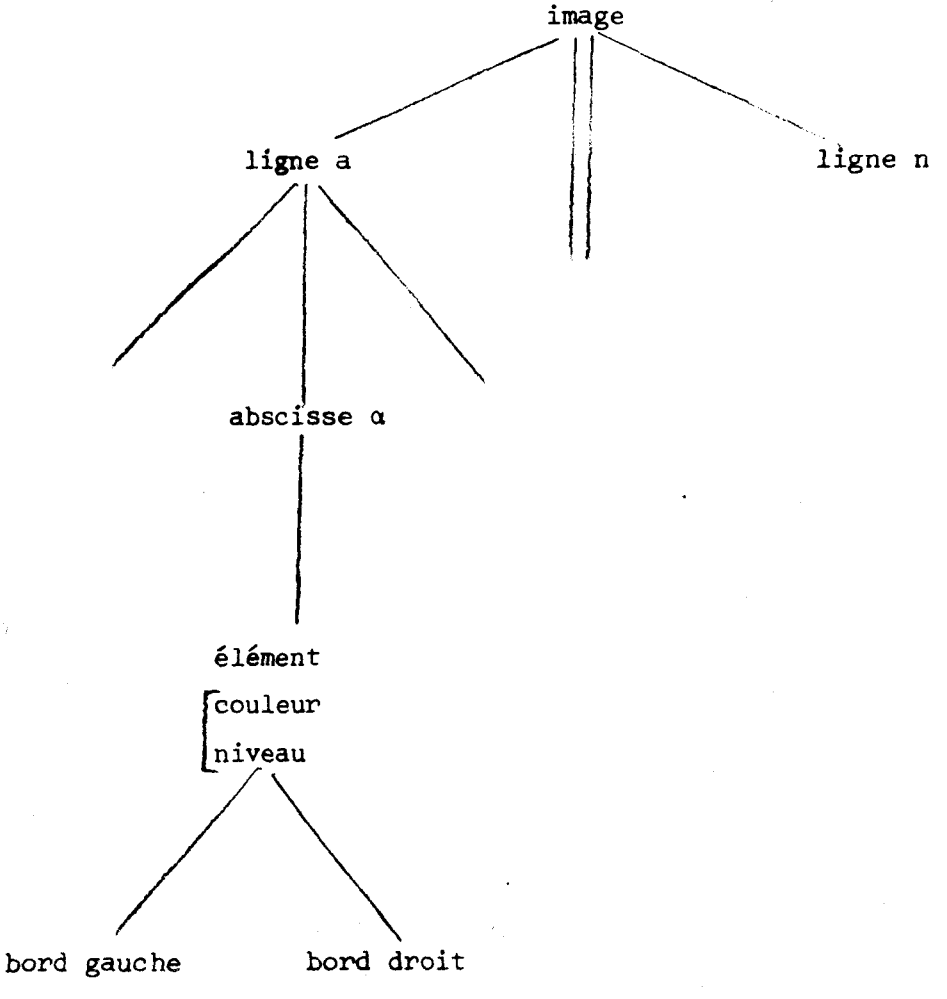
* associer à une tache les éléments décrits dans la mémoire.

Nous choisissons pour ce faire de caractériser une tache et les éléments la composant à l'aide d'un "nom" formé par

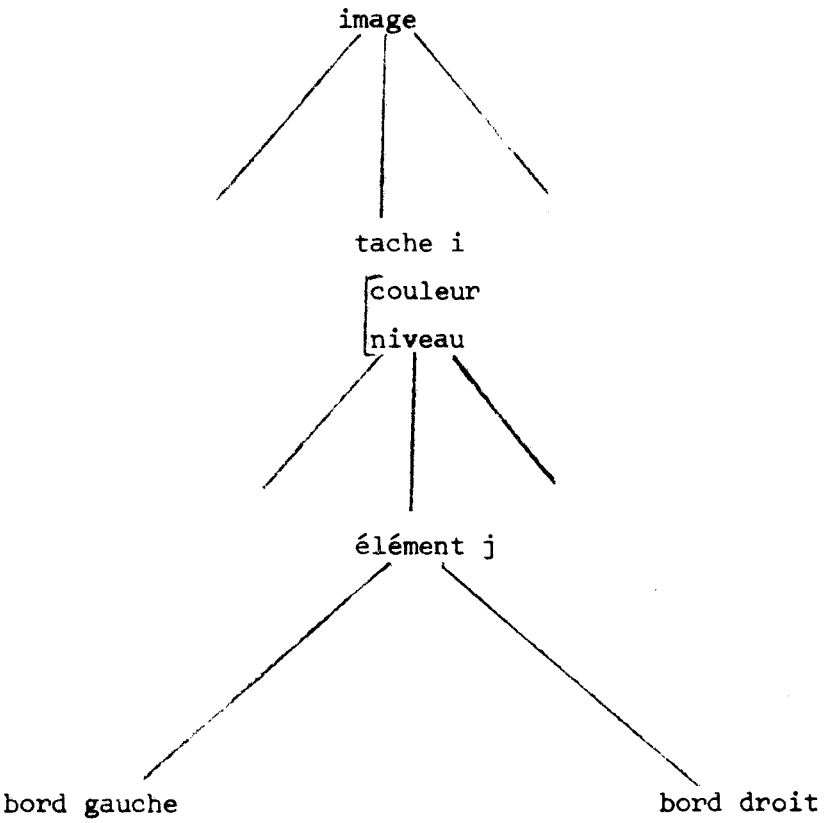
- la couleur
- le niveau (éventuel) où se trouve la tache

* ajouter et retrancher une tache dans cette structure.

Il y a donc deux structures "parallèles" qui facilitent : d'une part ces actions, d'autre part le décodage



Structure favorisant le décodage



Structure facilitant les actions de l'usager

Le problème est de réaliser facilement le passage de l'une à l'autre structure.

3-2-2-4 Réalisation

Elle sera étudiée en détail dans le § 4, et nous montrerons que la technologie actuelle ne permet pas de réaliser efficacement et complètement un tel processeur de façon uniquement matérielle. Nous nous sommes alors tournés vers une réalisation moins performante, traitant par logiciel une partie du problème.

3-2-3 Le processeur local ([Pal 76])

3-2-3-1 Choix

Le processeur local doit réunir les qualités suivantes :
- forte autonomie
- extensibilité

Nous nous sommes donc tournés vers un micro-ordinateur, dont nous pouvons maîtriser le fonctionnement au niveau le plus bas (bus, portes d'entrée-sortie notamment), et qui, tout en étant peu coûteux dans une version simple, permet une extension de configuration presque sans limite et de façon très modulaire.

3-2-3-2 Tâches lui incombant

Notre but n'est pas ici de développer un logiciel de haut niveau. Le processeur local à réaliser les tâches minimales suivantes :
- interface avec l'utilisateur
- interface avec les moyen de communication graphique
- interface universelle avec un processeur éloigné
- gestion de la structure logique de l'image, et réalisation d'actions de bas niveau, à savoir :
  - désignation d'un objet logique, considéré comme un ensemble de taches
  - création d'un objet
  - destruction [virtuelle (i.e. sur l'écran) / réelle (i.e. dans la mémoire)] d'un objet

Il peut également contenir un logiciel graphique élémentaire.

3-2-4 Le processeur éloigné ([Kri 62])

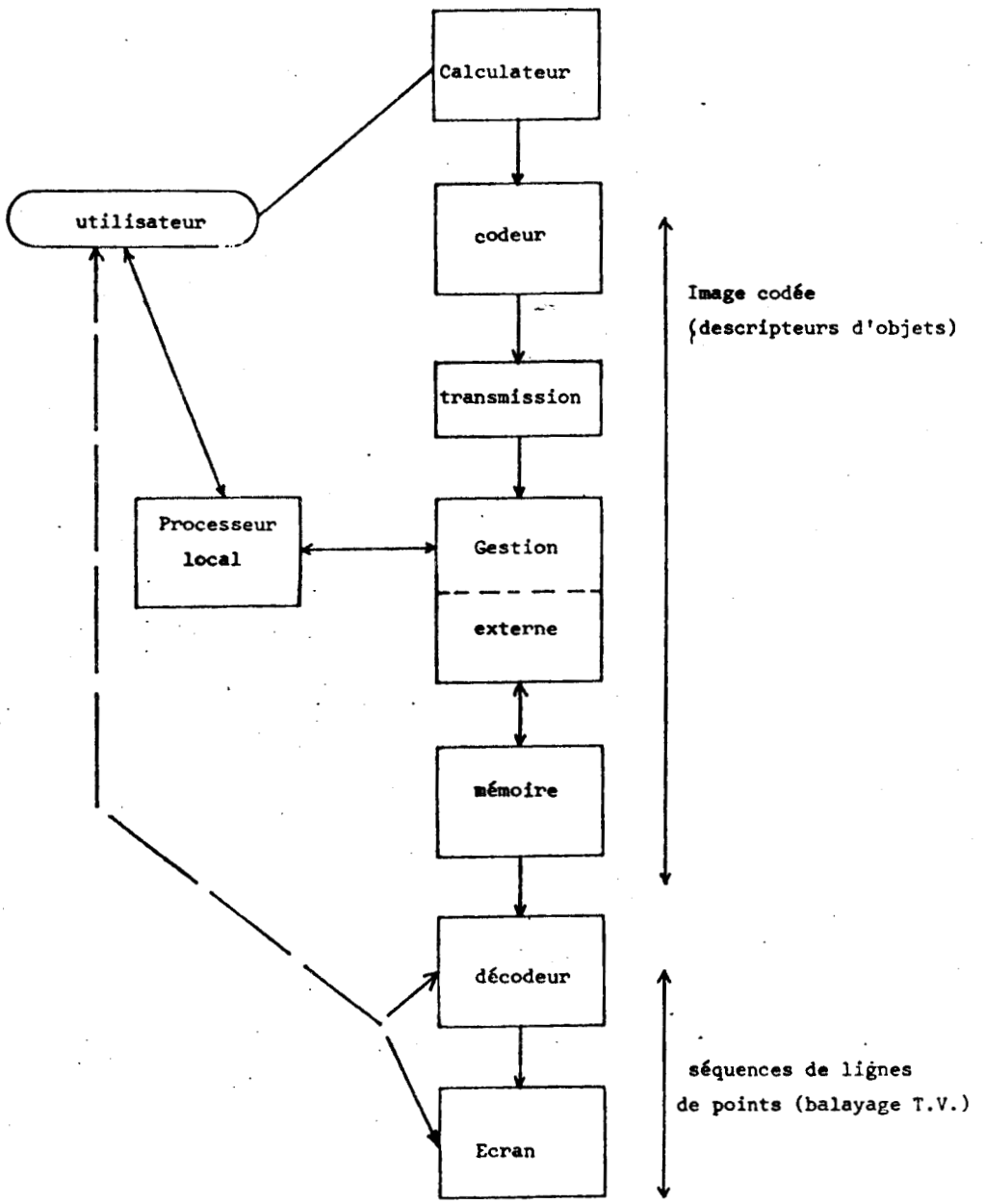
Nous disposons d'un gros ordinateur (IRIS 80), que nous pouvons connecter (par modem) à des vitesses allant de 11o à 4 800 bauds. Les interfaces matérielles et logicielles pouvant être standardisées, la connection à toute autre machine ou réseau est tout-à-fait possible.

3-2-5 Les moyens de communication

Le micro-ordinateur autorise pratiquement tous les moyens de communication, aussi bien les classiques moyens d'entrée sortie, que les moyens spécifiquement graphiques. Dans un premier temps paraissent indispensables :

- une télétype
- un curseur sur écran, associé au clavier ou à un photostyle ou à un "manche à balai".

3-3 Architecture globale.



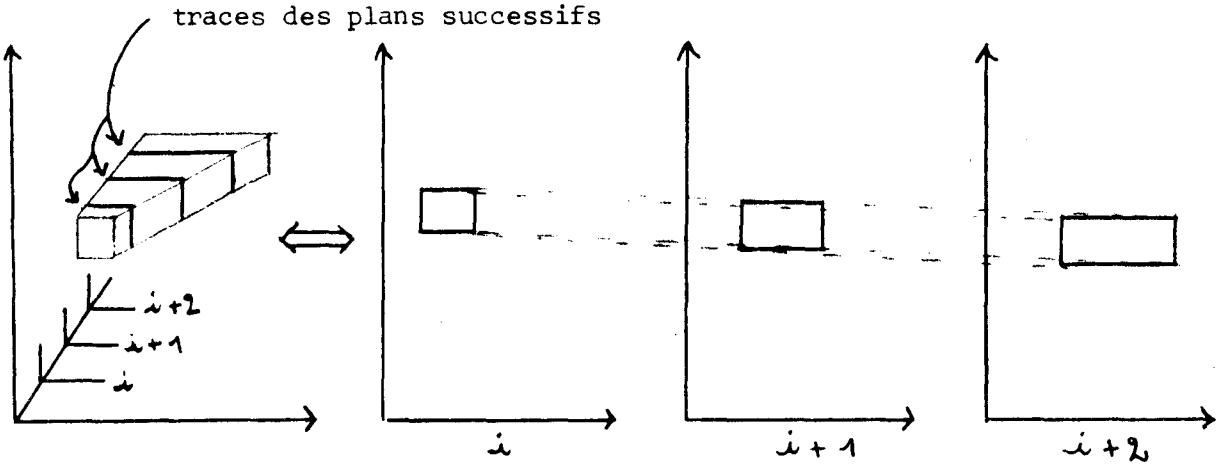
3-4 Performances attendues

3-4-1 L'écran

Comme nous l'avons vu, l'utilisation d'un téléviseur standard nous a conduit à choisir 256 lignes de 256 points. Nous avons de plus choisi de coder au niveau de l'écran la couleur sur 3 x 2 bits ; ceci n'est pas impératif : nous pourrions augmenter la taille, sous réserve d'utiliser un écran de meilleure qualité, ou le nombre de couleurs disponibles (ici 64) ; ce choix n'a aucune influence sur les autres éléments au système et peut être repris à tout instant.

3-4-2 Adéquation aux images

Comme nous ne mémorisons pas l'image sous forme matricielle, nous ne pouvons pas facilement représenter des images photographiques avant traitement ; mais il est clair, et le système a été conçu pour cela, que toutes les autres images sont faciles à représenter. De plus il prend en compte de façon locale une grande partie du problème des surfaces cachées, fréquemment rencontré en informatique graphique ; l'espace à trois dimensions classique utilisé alors peut en général se ramener à l'espace à plans parallèles que nous avons défini : un objet serait représenté par ses coupes dans les plans :



Ce problème n'a pas été étudié en son ensemble, mais on voit assez bien les avantages que l'on pourrait en retirer.

3-4-3 Adéquation aux actions et temps de réponse

Encore une fois, les choix que nous avons fait ne facilitent pas les actions globales ; certaines cependant sont réalisables simplement en utilisant un décodage judicieux : ce sont par exemple une translation ou une homothétie sur l'image.

Par contre les actions locales sont très efficaces : au niveau élémentaire, elles se ramènent à :

- créer une tache, c'est-à-dire rajouter ses éléments et leurs codes dans la mémoire
- modifier une tache, c'est-à-dire manipuler ses éléments (origines, codes, couleur, niveau)
- supprimer temporairement une tache, c'est-à-dire ne pas la décoder, mais la maintenir en mémoire
- supprimer effectivement une tache, c'est-à-dire dans la mémoire
- désigner une tache, c'est-à-dire effectuer la correspondance (x,y) visible sur l'écran <——> nom de la tache visible

Comme l'on travaille sur la mémoire de rafraîchissement, dès que le code est à jour le temps maximal de réponse est égal à la période rafraîchissement, à savoir 20 ms. Si le processeur local est suffisamment rapide, on peut donc faire du vrai dessin-animé.

3-4-4 Taux de compression - Mémoire de stockage

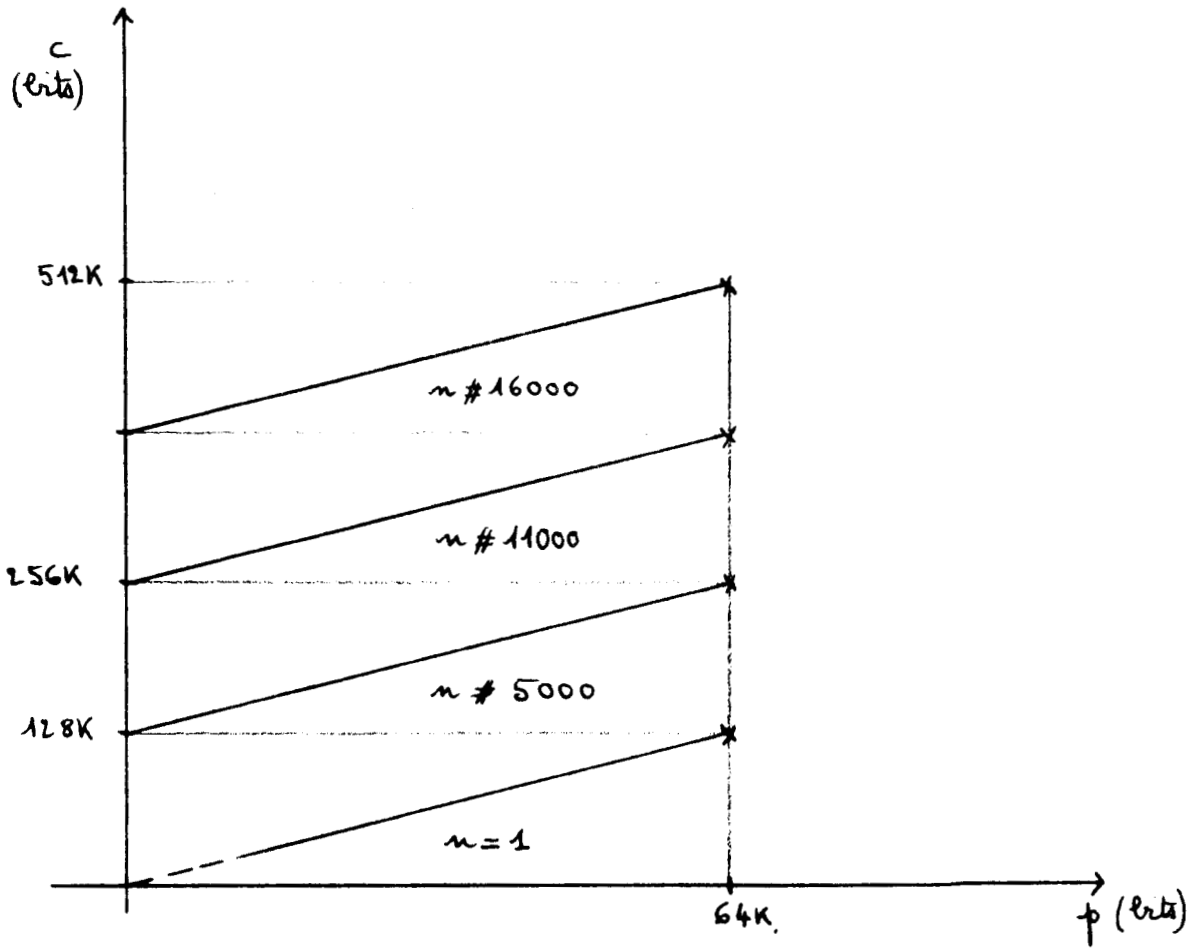
3-4-4-1 Dans le cas d'une image sans niveau, s'il existe n éléments dont la somme des périmètres est p^*, sans portions horizontales, la taille idéale du code serait

$$C_{bits} \simeq (8 + 8 + 8) \times n + 2 \times (p^* - n)$$

(8 + 8 : origine ; 8 : couleur ; 2 : code élémentaire ; $p^* - n$: origine)

soit $\boxed{c = 22\, n + 2\, p^*}$

p^*, dans ce cas, est égal à la complexité p de l'image, et l'on a pour une image 256 x 256 :



Si nous travaillons avec une taille de mémoire fixe, ce qui sera le cas en général pour un rafraîchissement, on a donc

$$22\ n + 2\ p^* \leq c_{max}$$

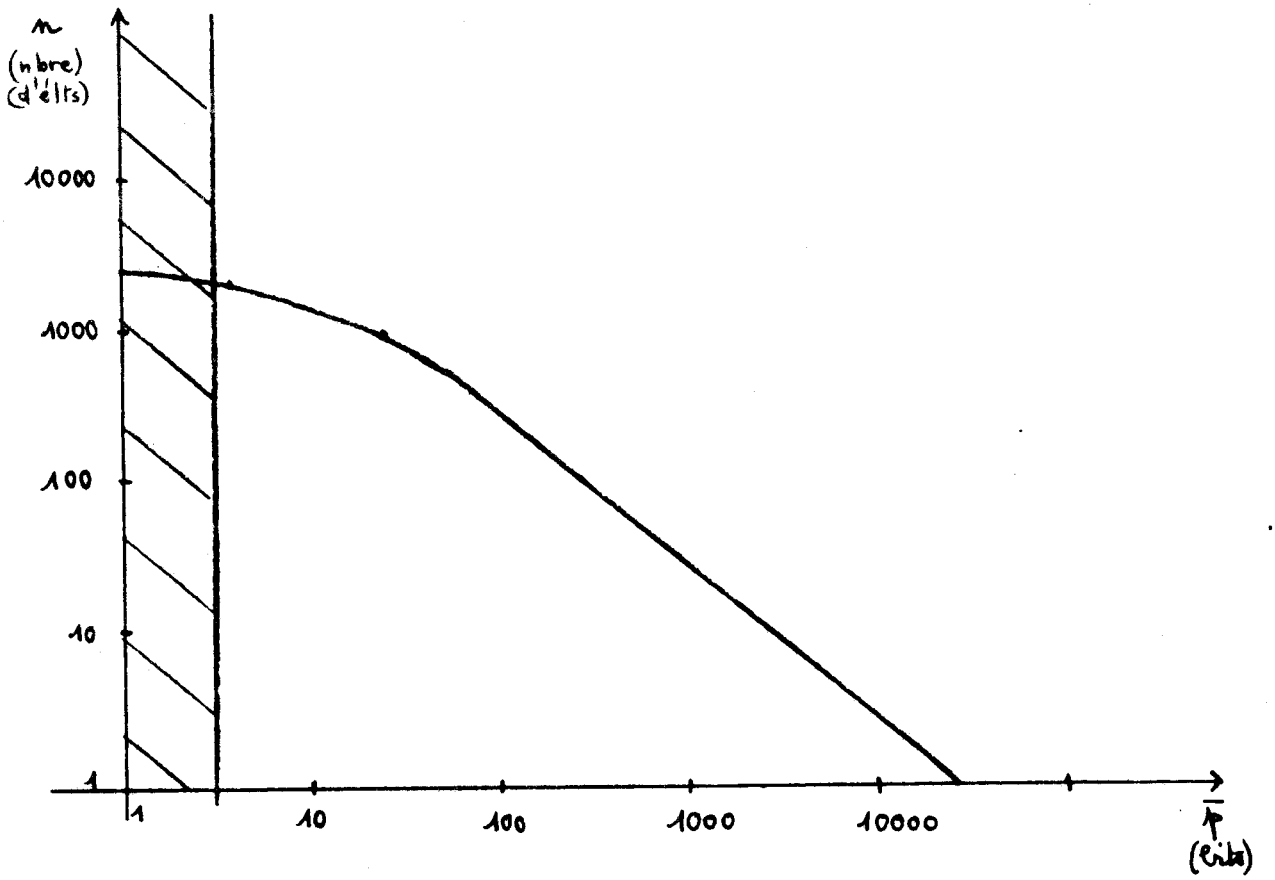
Si $\bar{p}$ est le périmètre moyen d'un élément, on a

$$\bar{p} = p^*/n$$

d'où $22\ n + 2\ n\bar{p} \leq c_{max}$

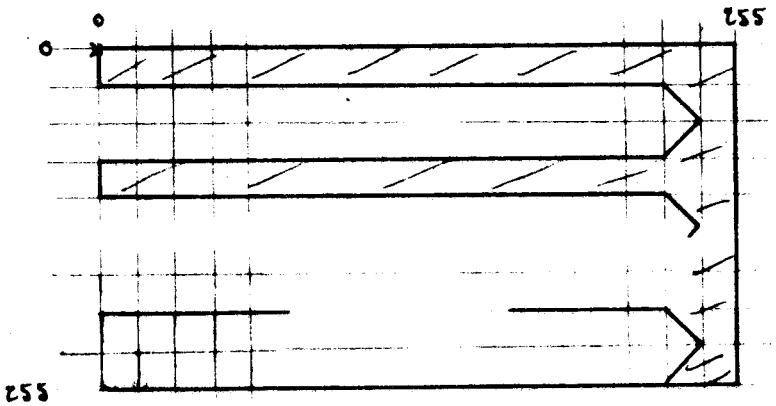
ou $\boxed{n\ (22+2\bar{p}) \leq c_{max}}$

Ainsi, si c_{max} = 64 Kbits, soit un taux de compression T_1 supérieur à 8, nous obtenons : $n\ (11+\bar{P}) \leq 32$ K.



Nous pouvons donc afficher

- un élément de périmètre 32 000 points. Nous ne pouvons donc afficher l'élément le plus complexe possible, qui est le "peigne" suivant,

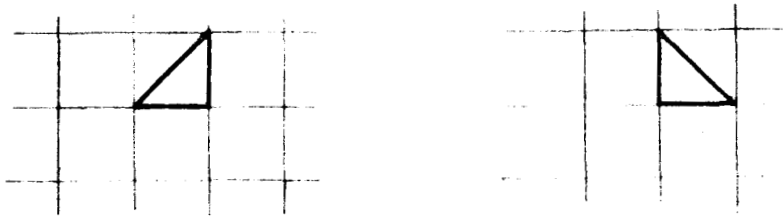


dont la complexité est voisine de

$$p \neq 256 \times \frac{2}{3} \times 256 \simeq 43 \text{ Kbits}$$

Mais ce cas est heureusement assez rare (!)

- environ 2340 éléments de périmètre (minimum) 3 bits, qui ne peuvent être que l'une des deux formes suivantes (car nous interdisons dans ces évaluations les portions horizontales en haut ou en bas de l'élément)



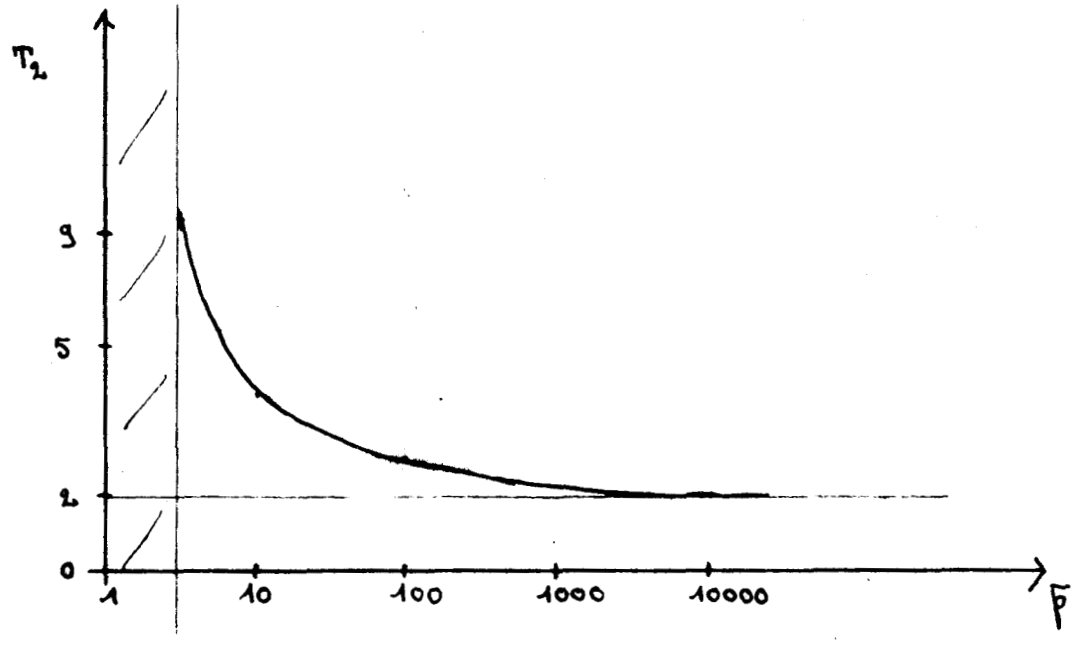
Une image uniquement formée de ces éléments pourrait en contenir 128 x 128, soit environ 16 000.

Mais il nous faut remarquer à nouveau que ces images sont assez exceptionnelles pour n'être pas prises en compte.

Nous pouvons maintenant évaluer le taux de compression T_2, qui est :

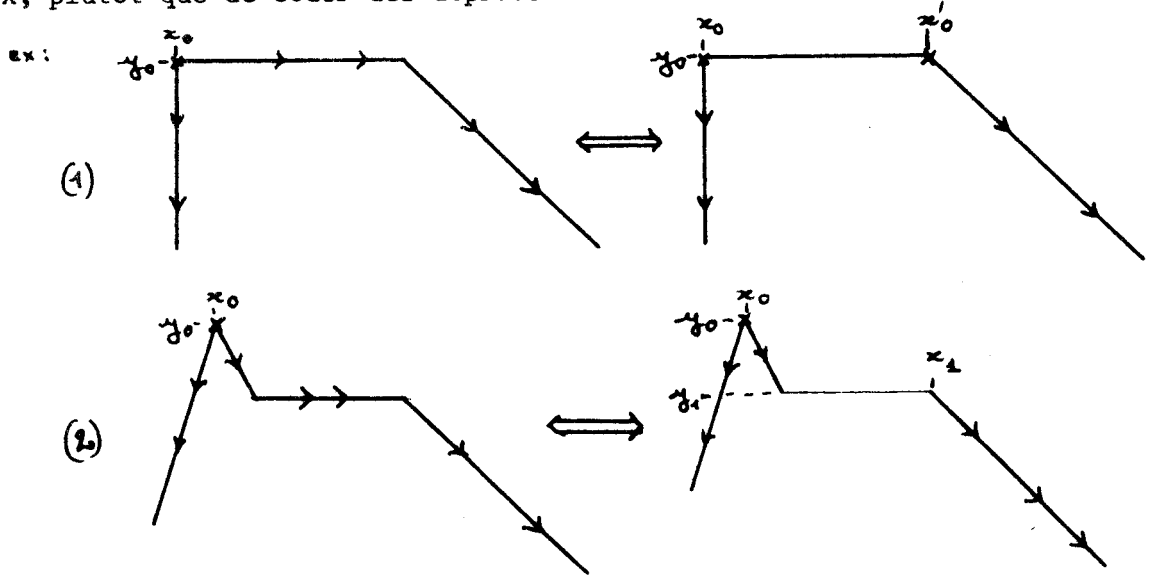
$$T_2 = c/p = \frac{22n + 2p}{p} = 2 + 22\ n/p$$

soit encore $T_2 = 2 + 22/p$

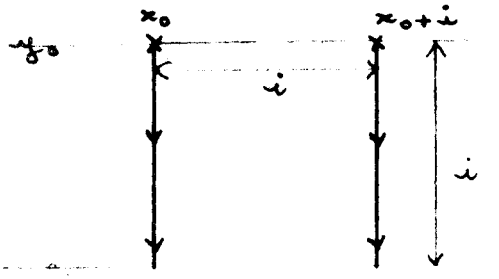


Nous mettons donc en évidence l'intérêt qu'il y a à travailler avec des éléments assez grands, dont le périmètre soit supérieur à quelques dizaines de bits.

3-4-4-2 Dans le cas d'éléments à portions horizontales, il convient de noter qu'il y a souvent intérêt à prendre une nouvelle origine en x, plutôt que de coder des déplacements unitaires



Développons le cas de l'exemple (1) dans le cas d'un carré :



on a alors p = 4i (complexité)

mais c_{bits} = (8 + 8 + 8 + 8) + 2 x 2i

- (8 + 8 + 8 + 8) : couleur
- y_o : code élémentaire
- x_o : code élémentaire
- x'_o : code élémentaire
- 2 x 2i : longueur à coder

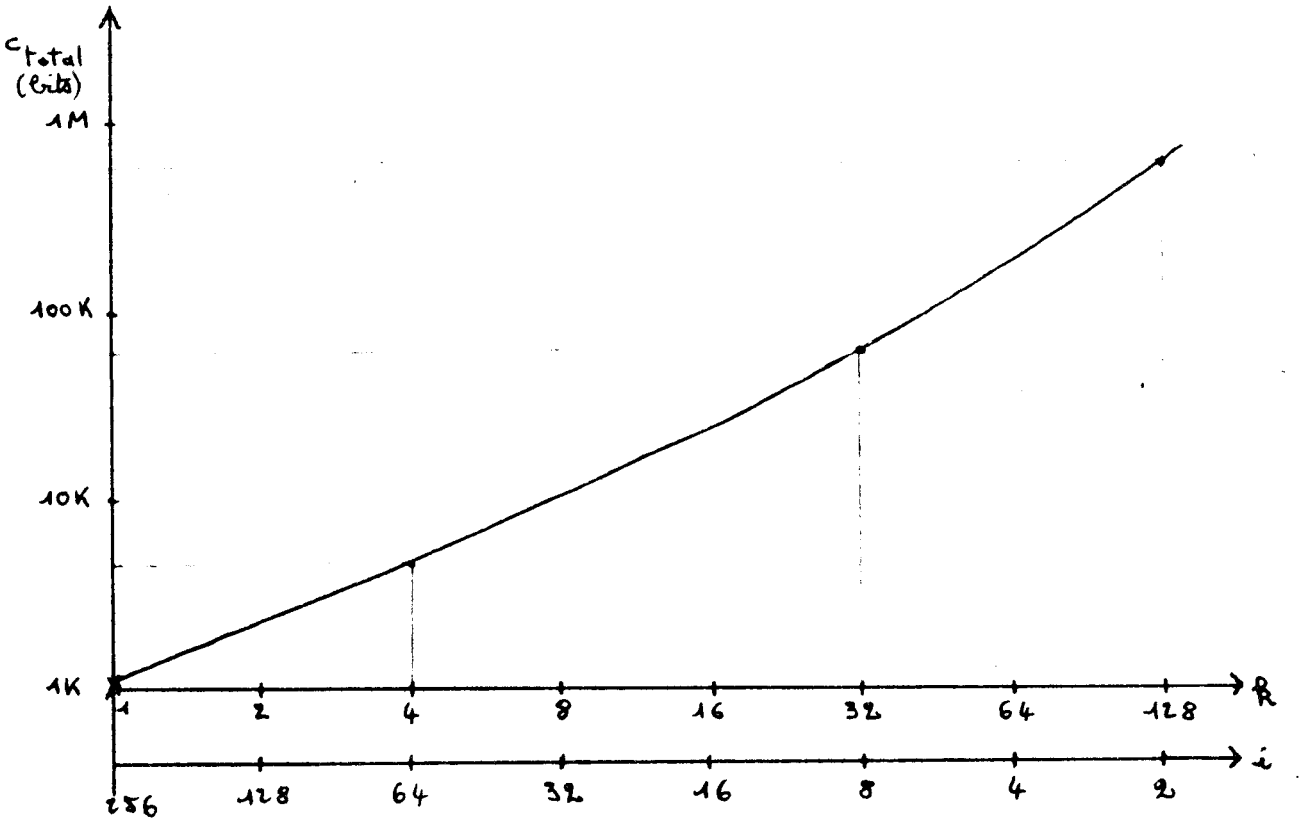
d'où (toujours pour 1 carré) :

$$\boxed{c = 32 + 4i}$$

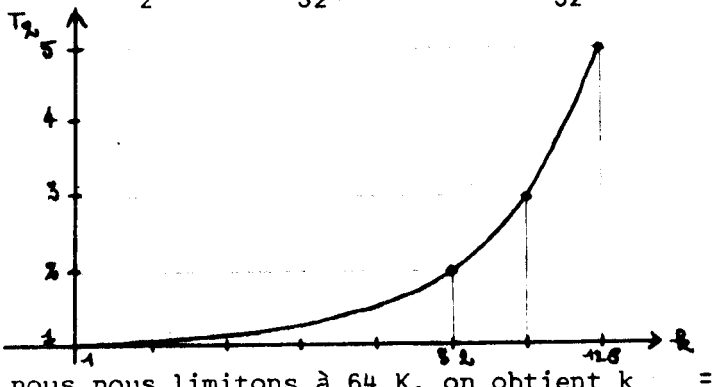
Si l'on divise l'écran 256 x 256 en k^2 carrés de côté i = 256/k, on aurait (sans astuce de codage) :

$$c_{total} = k^2 (32 + 4i) = k^2 (32 + \frac{1024}{k})$$

soit c total = 32 k (k+32) et p = 1024 x k.



on a donc également $T_2 = c/p = \frac{1}{32} (k+32) = 1 + \frac{k}{32}$,



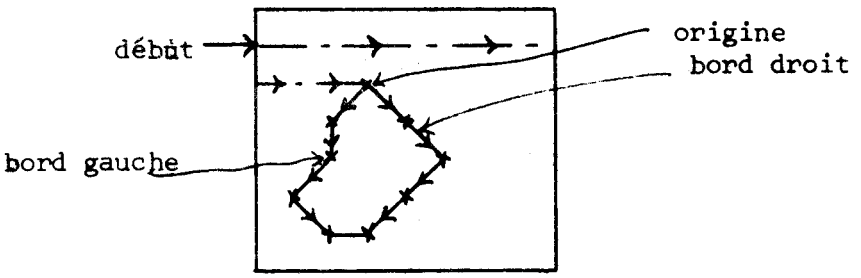
Si nous nous limitons à 64 K, on obtient $k_{max} = 32$, d'où $i_{min} = 8$ et $p_{max} = 32$ K ; de plus $T_2 \leq 2$

3-4-4-3 Il parait donc dans tous les cas raisonnable de se limiter à une mémoire de stockage de 64 Koctets, soit un taux de compression T_1 de 8, et un taux T_2 de quelque unités, ce qui est une bonne performance.

3-4-5 Codage et décodage

Dans la cas où l'image existe sous forme matricielle, il faut

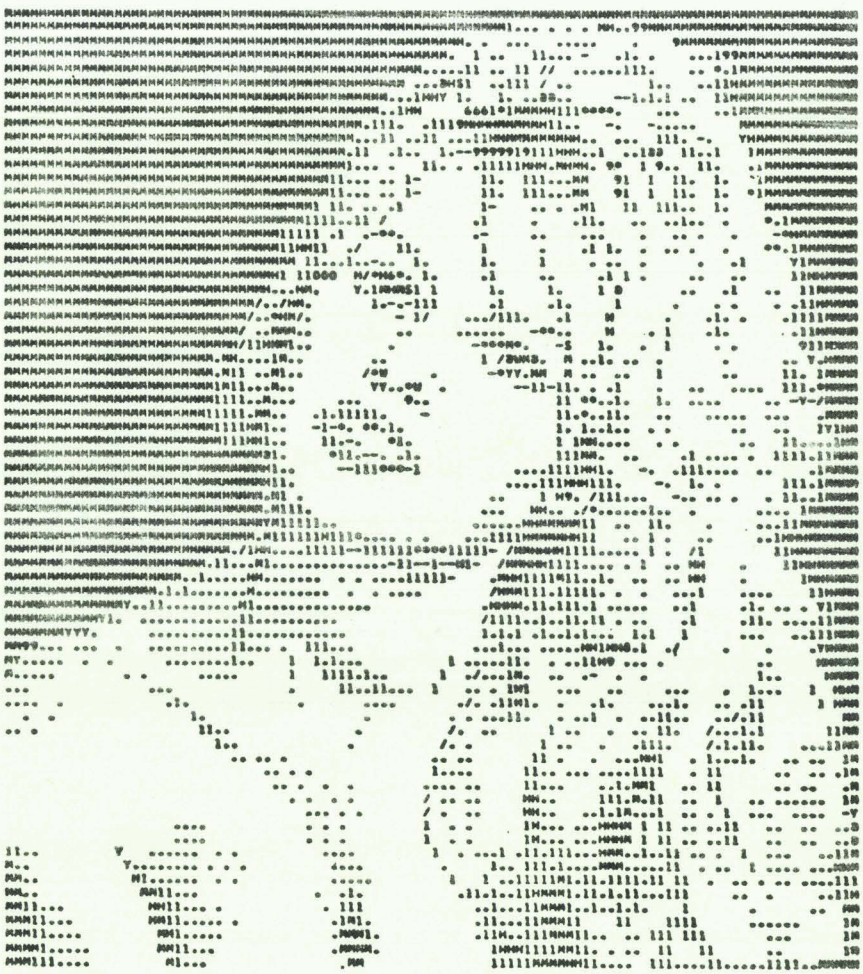
- analyser la matrice ligne par ligne de haut en bas, pour déterminer l'origine d'un élément donné
- suivre les bords ainsi trouvé (vers les y croissants) en élaborant le code au fur et à mesure



De tels algorithmes sont connus et assez faciles à mettre en oeuvre.

Si l'image est programmée ou interactive, il convient de prévoir le programme et le logiciel de façon à obtenir une description la plus proche possible de celle que nous désirons.

Le décodage est évidemment la partie la plus délicate du système, et une étude plus précise de sa faisabilité et de ses performances s'impose.



CHAPITRE 4

FAISABILITE DU PROCESSEUR GRAPHIQUE

4 FAISABILITE DU PROCESSEUR GRAPHIQUE

4-1 Introduction

Le processeur graphique comporte trois parties.

(1) *La mémoire d'image*, qui contient la description sous forme d'une structure de données contenant les informations suivantes :

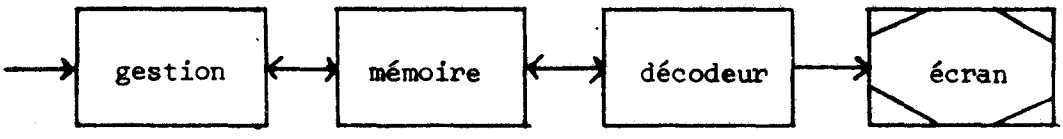
* pour chaque tache : sa couleur et son niveau
* pour chaque élément : son origine et deux séquences de codes décrivant les deux bords.

(2) *un dispositif de gestion de* cette mémoire, qui traduit dans cette mémoire les ordres élémentaires venant du processeur local, ordres que l'on peut résumer à

- créer une tache
- modifier une tache
- supprimer une tache

La structure des données contenues dans la mémoire d'image devra sinon faciliter, du moins autoriser la réalisation de tels ordres.

(3) *le décodeur*, chargé d'élaborer à partir de ces descripteurs l'image, c'est-à-dire la séquence de lignes et de points à afficher.



Nous nous intéressons ici à la réalisation du décodeur, et donc de la mémoire de descripteurs, car le dispositif de gestion est, de par sa nature (temps différé), moins difficile à réaliser ; nous ne perdrons cependant pas de vue le fait que celui-ci agit sur la mémoire de descripteurs, de façon aléatoire dans le temps et dans l'espace : dans le temps car cet accès peut intervenir a priori à n'importe quel moment, en particulier sans synchronisation avec le balayage de l'écran ; dans l'espace car cet accès peut concerner n'importe quelle tache, donc n'importe quel emplacement de la mémoire.

Nous pouvons imaginer plusieurs types de décodeurs :

. Le premier fournirait les séquences de points en temps réel, c'est-à-dire décoderait l'image mémorisée avec une période de 125 ns (durée d'un point). Ceci est évidemment absurde et irréalisable : il est clair qu'il faut élaborer l'image ligne par ligne et non point par point.

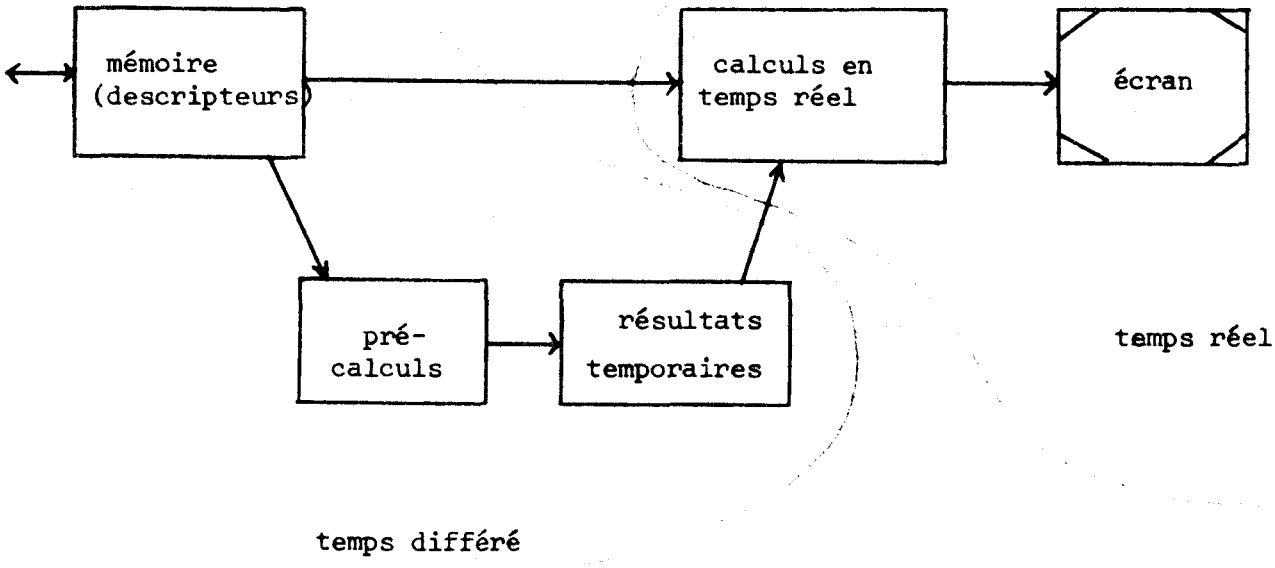
. Un deuxième type consisterait à décoder l'image (ligne par ligne) à l'aide d'un monoprocesseur, élaborant séquentiellement les résultats nécessaires à la construction d'une ligne. Nous montrerons que ce processeur est actuellement irréalisable.

. L'étude de ce monoprocesseur nous amènera à une architecture parallèle (pipeline) où le décodeur est scindé en plusieurs opérateurs reliés par des files d'attente.

. Ce système, théoriquement faisable, nous conduira à associer à chaque niveau de profondeur un processeur de structure analogue ; nous obtenons donc n processeurs "pipe-line" simples, identiques, fonctionnant en parallèle.

. Une autre façon de paralléliser et de modulariser le système consisterait à associer à chaque tache, voire à chaque élément, un processeur spécialisé simple. Ce procédé permettrait une extensibilité quasi-infinie, mais n'est envisageable que si le processeur est complètement intégré à raison d'un ou mieux de plusieurs sites par boîtier.

. Une dernière catégorie concerne les réalisations semi-différées : il s'agit, à partir de la description de l'image, de calculer en différé, une fois pour toutes, certaines données de l'image. La partie "temps réel" du décodeur utilise ces résultats et/ou les descripteurs d'image pour élaborer l'image.



Nous étudierons une solution où l'analyse des niveaux est différée, et une autre où le décodage des contours est différée. C'est cette dernière qui a été effectivement réalisée.

4.2 Architecture monoprocesseur

Nous supposons ici que nous disposons d'un seul processeur qui élabore ligne après ligne les points à afficher. Il exécute donc cycliquement un algorithme consistant à :

(1) calculer pour chaque bord traversé par la ligne en cours l'abscisse nouvelle en fonction de l'ancienne et du code. S'il y a N_c éléments, il faut donc appliquer $2N_c$ fois cet algorithme.

(2) mémoriser les données nécessaires aux calculs de la ligne suivante, c'est-à-dire pour chaque bord, l'adresse du code en cours et l'abscisse actuelle.

(3) introduire les nouveaux éléments à prendre en compte

(4) élaborer la ligne à afficher à l'aide des triplets (abscisse, couleur, niveau) ainsi calculés.

Cherchons à évaluer d'une part la taille de la mémoire de stockage des données temporaires, d'autre part le nombre d'opérations élémentaires à effectuer :

(1) mémoire temporaire.

S'il y a N_c éléments à prendre en compte, il faut mémoriser deux abscisses (2x8 bits) et deux adresses de code (environ 2x16 bits) par élément, soit environ $48xN_c$ bits. Il faut aussi mémoriser les résultats pour l'affichage, soit N_c fois le triplet

- abscisse = 8 bits
- couleur = 8 bits = 24 N_c bits
- niveau = 8 bits

Il faut donc environ 72 N_c bits de mémoire. Si l'on veut par exemple pouvoir afficher 64 éléments (coupés par la même ligne), il faudrait environ 4600 bits de mémoire.

(2) nombre d'opérations.

La phase (1) nécessite 2 N_c x (nombre d'opérations de l'algorithme). Dans le cas le meilleur, l'algorithme s'exécute en 5 opérations

(cas ou code = 1 et code suivant ≠ 3)

```
1   lire code ;
2   si code = 1
3       alors faire x ⟵ x - 1
4                   lire code
5                   tant que c = 3 ...

        sinon . . .

    fin
```

Il faut donc 10 N_c opérations, avec en plus 2 N_c lectures de code et 2 N_c lectures d'anciennes abscisses, soit 4 N_c lectures.

La phase (2) demande 4 N_c écritures. La phase (4) consiste en une suite de comparaisons, que l'on peut évaluer à N_c. Il faut donc environ 11 N_c opérations élémentaires et environs 8 N_c lectures-écritures. Pour N_c = 64, il vient environ 700 opérations et 510 lectures/écritures : si le fonctionnement est séquentiel, on obtient un temps moyen alloué à chaque opération ou lecture-écriture de

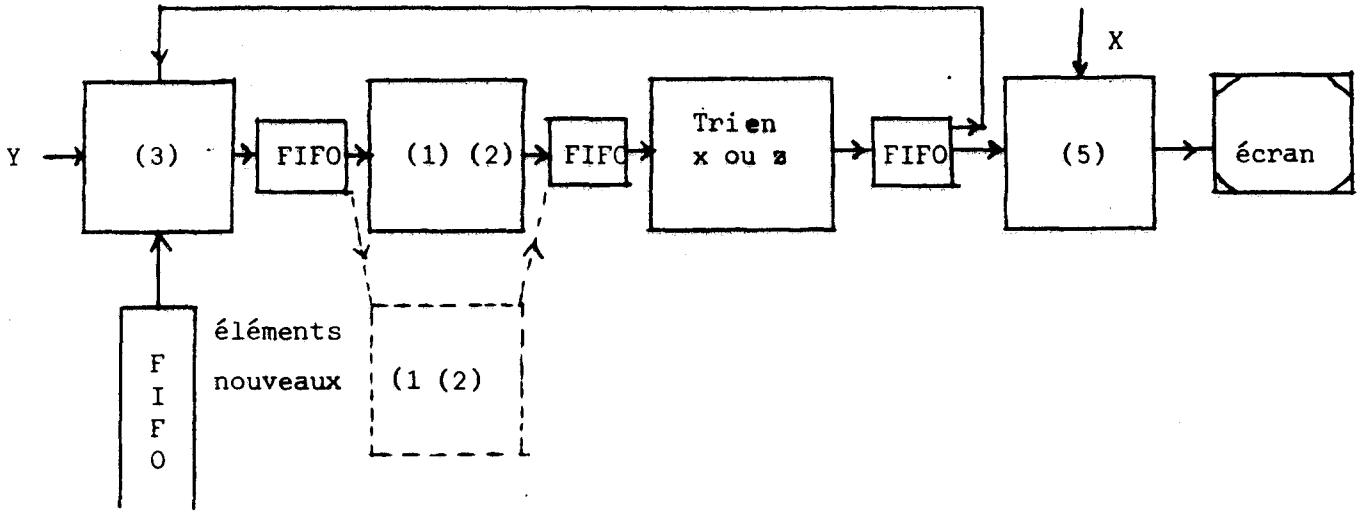
$$64\ \mu s\ /\ (700 + 510) \simeq 50\ ns$$

Mais l'algorithme peut être (beaucoup) plus long ; de plus on n'a pas tenu compte de certaines manipulations intermédiaires (indirections, générations d'adresses). Si l'on multiplie par 2 les estimations, on obtient un temps moyen alloué de 25 ns : ceci n'est pas réalisable sur un processeur ou une mémoire de technologie usuelle ; il faudrait utiliser des technologies ultra-rapides (I^2L, ECL), avec en contrepartie des difficultés de réalisation (bruit, connexions...), une consommation élevée, et un coût très élevé.

En conclusion, il paraît impossible de réaliser le décodeur de cette façon. Néanmoins cette analyse a mis en évidence différentes phases du décodage et leur importance relative.

4-3 Architectures pipe-line

4-3-1 Les phases définies ci-dessus sont parallélisables à la condition que la phase (4), qui est une phase de tri, puisse se faire par simple permutation. Il convient donc de trier les résultats de calcul soit en abscisses, soit en niveaux. On peut alors adopter la structure "pipe-line" file d'attente suivante :



(1) (2) : décodeur

entrées : * ancienne abscisse
* adresse du code en cours

sorties : * nouvelle abscisse
* résultat
* adresse du code en cours

fonction: * réaliser l'algorithme de décodage des contours.

(3) mise à jour

entrées : * file d'attente des éléments à prendre en compte, triés en y_o croissants (et x_o croissants)
* y de la ligne en cours
* z ou x des éléments déjà pris en compte

sorties : * z ou x des nouveaux éléments
* adresse du code en cours pour ces éléments

fonction: * introduire les éléments au bon endroit

(4) tri

entrées : x

sorties : x

fonction: rétablir l'ordre (en x) des données

On peut en effet imaginer deux types de fonctionnement : l'un en x : les données sont dans la boucle en x croissants ; le rôle du comparateur (5) est alors, en fonction du x courant, de déterminer quelles sont les couleurs existantes et sur quel niveau ; il décide alors de la couleur à afficher (par exemple, à l'aide d'une pile). L'autre en z : les données sont rangées en niveaux croissants, et dans chaque niveau en x croissants. Le comparateur procède à une projection de ces lignes des différents plans sur la ligne qui sera finalement visualisée.

Nous pouvons préciser la structure des données dans les files d'attente de la boucle : il faut mémoriser pour chaque bord

* la couleur 1 octet
* le niveau 1 octet

* la nature (gauche ou droit) 1 bit
* l'ancienne abscisse 1 octet
* l'abscisse résultat 1 octet
* l'adresse du code en cours 2 octets

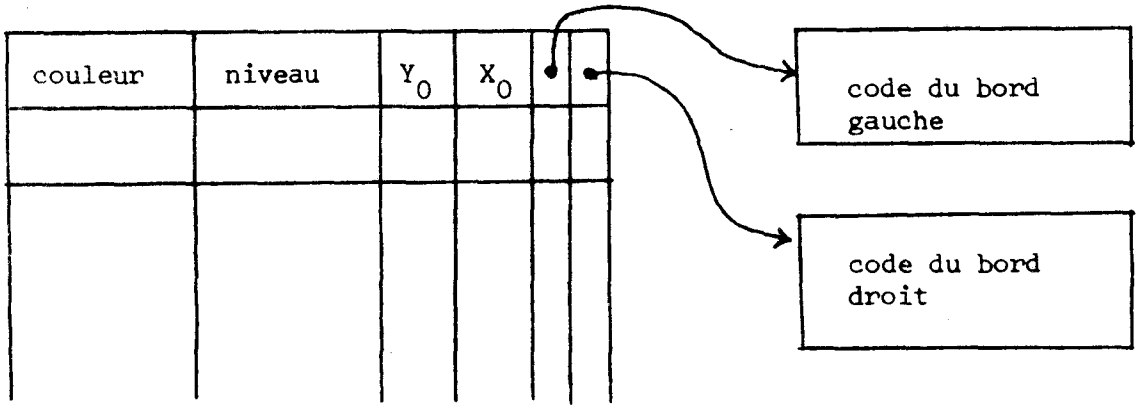
total $\sim$ 6 octets

Les éléments nouveaux doivent se présenter en y d'origine croissants, et suivant le cas

* en y (et x) croissants
* en x (et z) croissants

Il paraît donc intéressant de structurer la mémoire de descripteurs de la façon suivante :

* une table des origines et noms des éléments, triée en y croissants, et x ou z croissants
* un ensemble de code



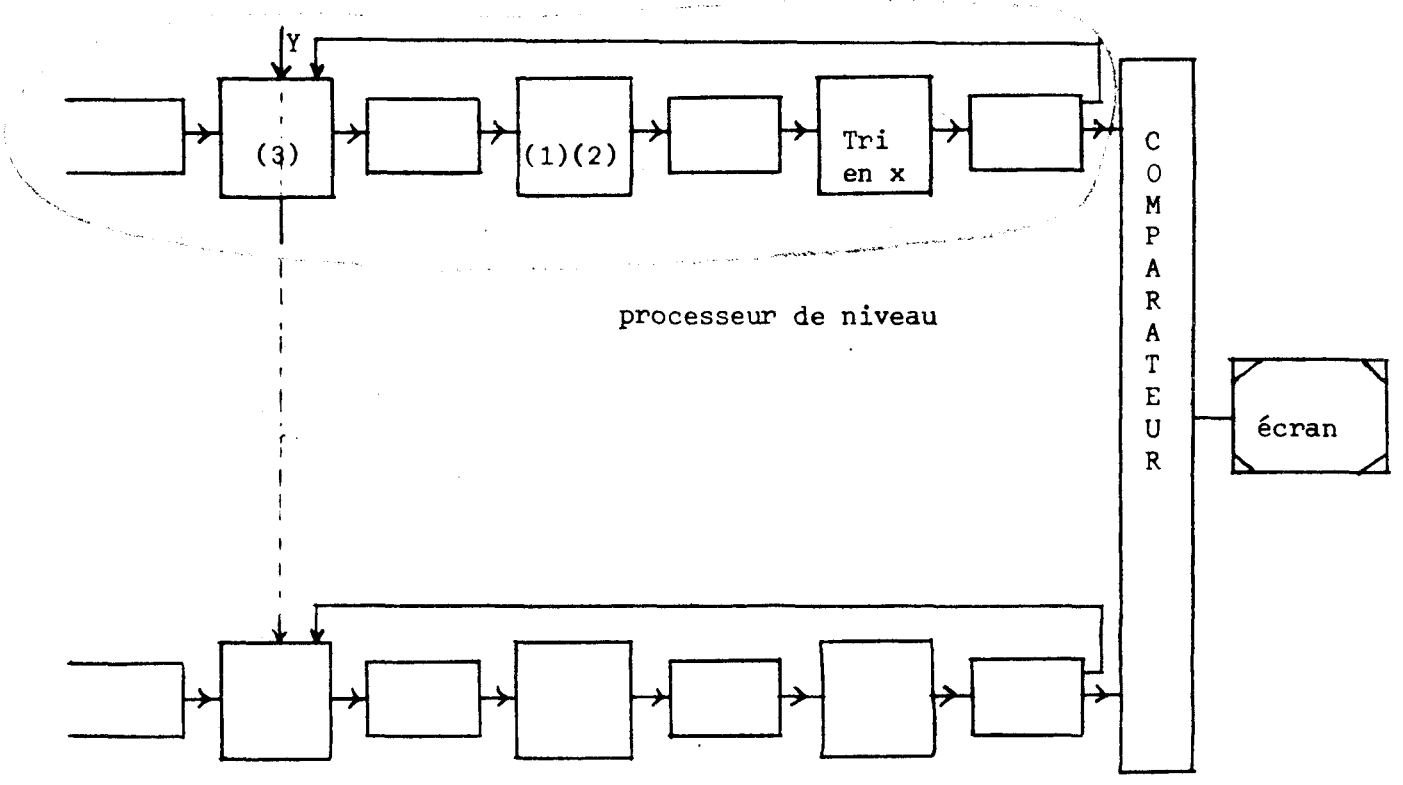
Nous n'allons pas détailler plus avant cette architecture ; mais précisons simplement les points critiques, qui sont :

- les performances du décodeur de contours ((1)(2)), qui demande pour fonctionner au moins un accès mémoire, et au moins 5 opérations élémentaires par bord. Il y aurait donc lieu d'en créer plusieurs fonctionnant en parallèle (les bords peuvent en effet se succéder sur 2 points voisins).

- la taille et les performances des files d'attente : pour autoriser 64 éléments, il faudrait environ au total 128 positions de 6 octets, susceptibles de se décaler en moins de 500 ns.

- l'opérateur (5) d'élaboration de la couleur en fonction du x courant et d'une file d'attente de (x,z, nature du bord). Le nombre de comparaisons à effectuer peut être élevé et empêcher le fonctionnement en synchronisme, qui impose un cycle de 125 ns ; cet opérateur peut avoir jusqu'à 256 données à comparer. Une solution différée sur une ligne est envisageable, mais également malaisée.

4-3-2 Ce dernier point et la redondance d'informations de niveau nous conduit à associer une telle structure simplifiée à chaque niveau en complétant le dispositif par un comparateur.



Ce dispositif permet de simplifier et de banaliser un processeur de niveau ; le système devient donc extensible de ce point de vue. Il est ici intéressant de structurer la mémoire de descripteurs en :

- une table des origines et noms triée en z croissants, puis en y d'origine croissants, puis en x d'origine
- une mémoire de code

Dans ces deux architectures, le processeur de décodage (1)(2) est obligatoirement asynchrone. Il faut donc disposer de vraies files d'attente ; la technologie actuelle ne fournissant pas de telles fonctions de façon satisfaisante (taille, vitesse, coût), nous n'avons pas retenu cette solution.

4-4 Architectures multi-processeurs

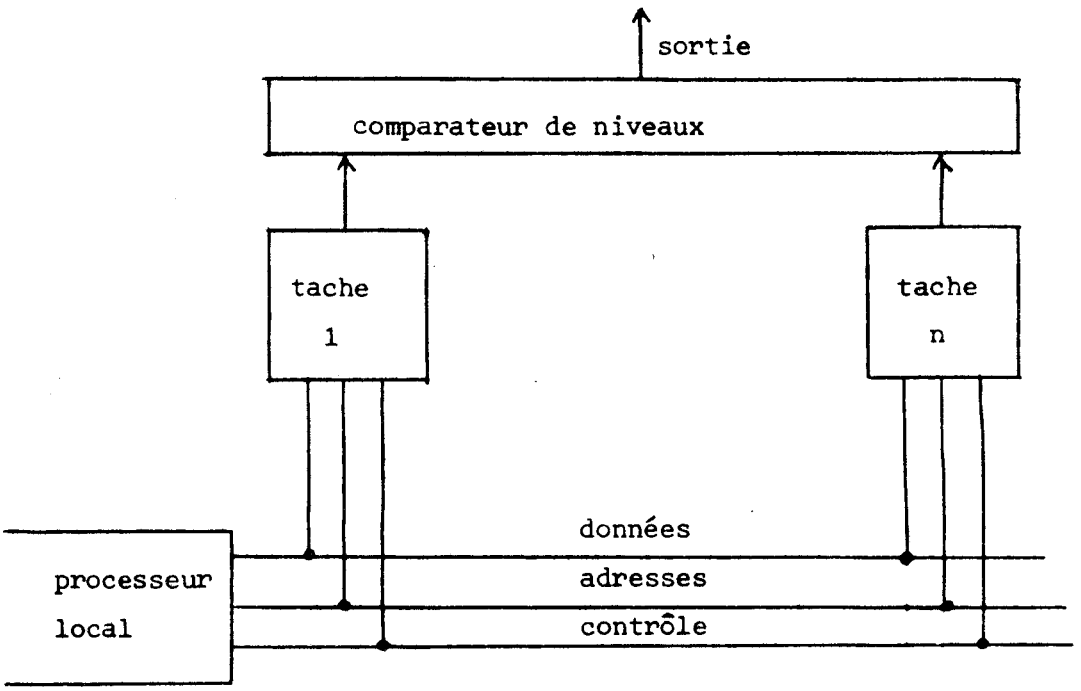
Hormis les architectures pipe-line décrites ci-dessus, nous pouvons imaginer une structure différente, complètement parallèle, où l'on associerait un processeur non plus à chaque niveau, mais à chaque tache, c'est-à-dire à chaque objet manipulé par l'utilisateur. Ce processeur doit

—> mémoriser les données relatives à chaque tache, à savoir :
  * son nom, composé du niveau et de la couleur
  * ses divers éléments, caractérisés par la donnée de l'origine et des deux bords

—> être accessible par le processeur local par simple présentation d'une adresse, c'est-à-dire de son nom sur un bus

—> effectuer en temps réel le décodage des descripteurs de la tache qui lui est associée

Il reste alors à disposer d'un comparateur de niveaux comparable à celui de l'architecture précédente pour élaborer la séquence de points à visualiser.



Essayons de préciser l'architecture d'un "processeur-tache".

1) Accès par le processeur local :

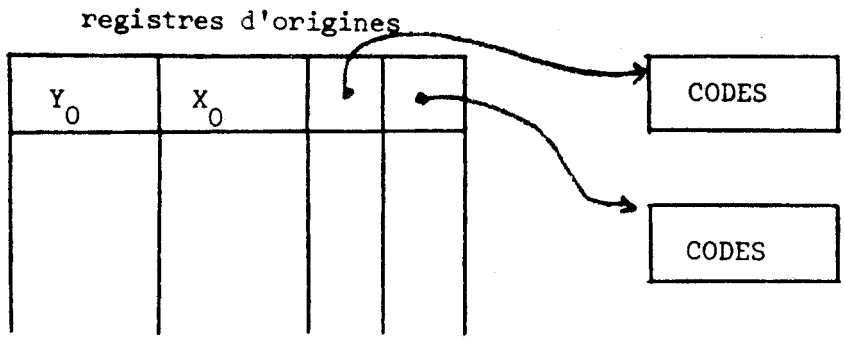
Chaque processeur doit se reconnaître si le processeur local présente son nom (couleur + niveau) sur le bus d'adresses (accès associatif). Il suffit de le doter d'un registre d'adresse contenant son nom, d'un comparateur, et également d'une bascule d'état (occupé, non occupé) indiquant l'allocation du processeur.

- pour créer une tache, on cherche un processeur inoccupé, que l'on active. Il suffit alors de ranger son nom dans le registre de nom, puis d'écrire dans sa mémoire les données relatives à cette tache, puis d'activer la bascule d'occupation

- pour adresser une tache, dans le but de la modifier, de la détruire, etc..., il suffit de présenter son nom sur le bus. Elle se sélectionne automatiquement.

2) Mémoire locale :

Il faut mémoriser les éléments constitutifs de la tache, par exemple sous forme d'une liste classée en ordonnées d'origine croissantes (et en abscisses d'origines croissantes), chaînée vers la suite des codes décrivant les contours.



On peut fixer une taille maximale

- pour le code, quite à créer un nouvel élément si la taille est trop élevée
- pour les registres d'origines, quite à associer une tache à plusieurs processeurs (moyennant des précautions d'accès par le processeur local)

3) Décodage :

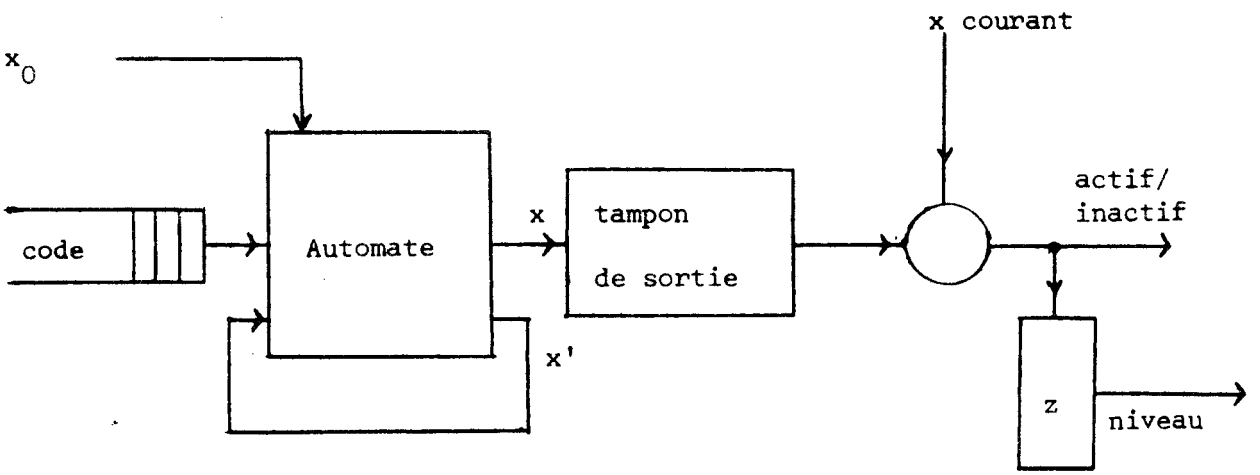
Il faut simplement ici décoder les codes des bords, ligne par ligne. Il s'agit d'un automate ayant comme

- entrée initiale : x_o
- entrée courante : le code (ou son adresse)
- entrée mémorisée : x_a, résultat du cycle précédent
- sorties :
  - x réellement affiché
  - x_a, résultat servant au cycle suivant.

Cet (ou ces) automate(s) fournit (ssent) une séquence triée (implicitement, car on ne traite qu'une tache) de couples (x, nature du bord). En fonction du x courant de l'écran, la sortie sera

- inactive si l'on est hors d'un élément

- active si l'on est dans un élément dans ce cas, on présente le niveau z de la tache en sortie.



Il ressort clairement de cette description qu'il n'est pas réaliste d'envisager une implémentation discrète d'un tel processeur : il faudrait au minimum 10 boîtiers par processeur, soit plusieurs milliers si l'on veut une machine évoluée.

Cependant, si l'on parvient à intégrer un tel processeur, voire plusieurs dans un même boîtier, cette architecture n'est plus "inhumaine", et à l'avantage d'une extensibilité simple et évidente ; la technologie nous permet d'espérer la réalisation d'un tel processeur "sur-mesure" dans un délai de quelques années.

Disons quelques mots du comparateur final. Ses entrées sont :

- des bascules d'activation des taches
- des valeurs de niveau pour les taches activées

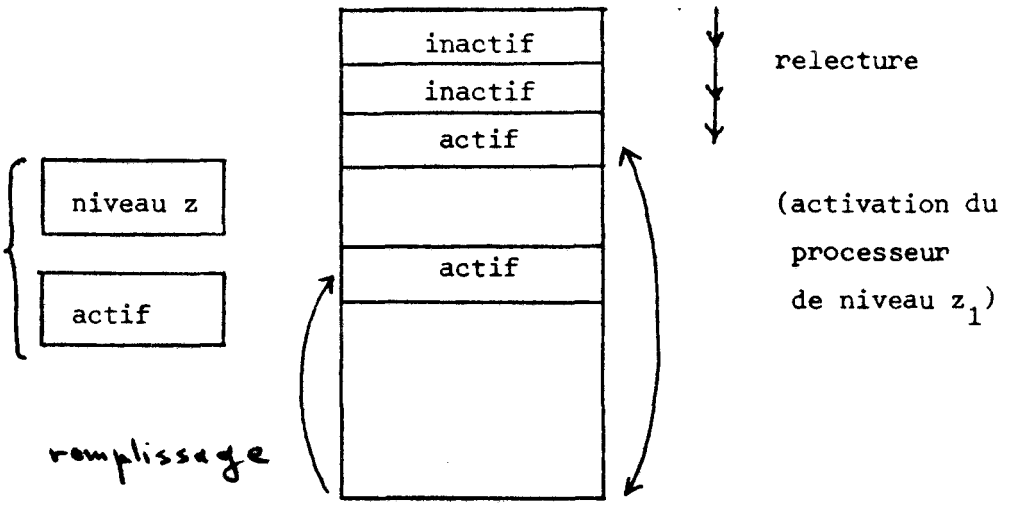
Il doit activer en retour la tache sélectionnée pour l'affichage, c'est-à-dire celle du niveau le plus proche de l'écran. On peut différencier un fonctionnement temps réel, au niveau du point, où le calcul doit se faire en 125 ns, d'un fonctionnement différé au niveau de la ligne, où l'on peut tolérer une méthode moins efficace. Nous pouvons indiquer plusieurs solutions, à savoir

- le balayage de z = 0 à z = 255 avec un système de comparateurs : on s'arrête dès que l'on a trouvé un processeur actif sur le niveau observé.

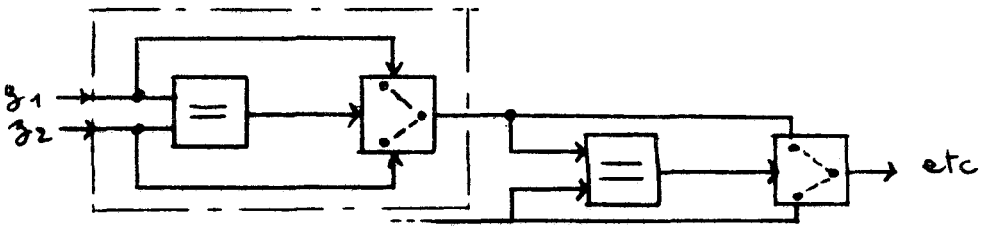
- le balayage des processeurs pour trouver lequel est actif sur le niveau le plus proche. (ce balayage doit être exhaustif)

- un procédé analogue au système de report rapide de retenue ; un processeur actif sur le niveau n désactive tous les processeurs actifs de niveaux inférieurs

- une gestion par "pile" ; on crée une pile de niveaux, que l'on remplit par le processeur, de façon aléatoire, et dont on relit le sommet



- un système à multi-comparateur (analogue à la méthode de tri dite du "tournoi")



(pour 2^n entrées, il faut N = 2 - 1 couples comparateur-multiplexeur ; ainsi, si n = 8 (soit 256 taches), N = 127)

Ici encore, ces solutions ne sont envisageables que dans le cadre d'une intégration assez poussée.

En conclusion, nous remarquerons simplement les commodités qui résulteraient d'une telle association tache de l'image <——> processeur matériel : modularité, extensibilité (jusqu'à 32 K taches !!), performances (recherche des taches par le nom, vitesse, représentation sémantique de l'image).

4-5 Architectures semi-différées

Nous avons mis en évidence deux aspects du décodeur, que l'on peut séparer complètement, à savoir

- le décodage des contours
- le traitement des niveaux

Nous pouvons donc envisager de différer l'une de ces deux fonctions, c'est-à-dire de l'effectuer de manière asynchrone, une fois pour toutes. Nous obtenons donc deux architectures semi-différées.

4-5-1 Gestion différée des niveaux

Il s'agit à partir des descripteurs des taches dans l'espace à plans parallèle, d'obtenir des descripteurs de taches disjointes dans le plan de l'écran. Nous pouvons alors poser deux problèmes

- il est probable que, par superposition, les taches résultantes auront de nombreux bords communs. Faut-il répéter les descripteurs correspondant ou les chaîner ?

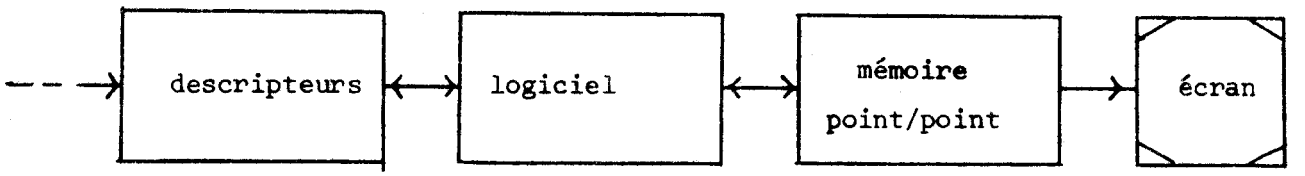
- la superposition de deux taches initiales sur des plans différents implique de pouvoir affirmer

  (1) - si 2 taches sont ou non totalement ou partiellement superposées

  (2) - quel est le code résultant pour une tache cachée partiellement

Au problème (1) sont associés deux sous-problèmes
- commant sait-on que 2 bords se coupent ?
- comment sait-on qu'une tache est intérieure à une autre ?

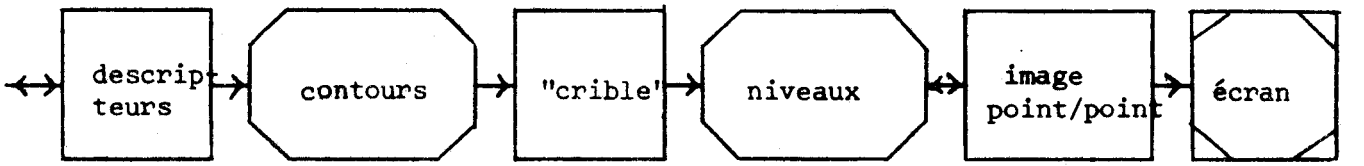
Ces problèmes n'ont pas de solution simple, si ce n'est dans le cas où l'on se reporte à un espace de travail matriciel, c'est-à-dire où l'on effectue le décodage des contours. Il n'y a alors pas lieu de recoder l'image, et l'on obtient une solution complètement différée :



Cette solution est tout-à-fait réalisable, mais présente moins d'intérêt pour le but que nous poursuivons.

4-5-2 Décodage différé des contours

Il s'agit ici de décoder les contours un par un, une fois pour toutes ; il faut donc mémoriser point par point les contours calculés. Un processeur de niveau effectue alors l'intégration dans l'image finale.



L'existence d'une copie de l'écran en mémoire apparaît comme indispensable. De plus cette mémoire doit contenir en chaque point le niveau visible (et sa couleur).

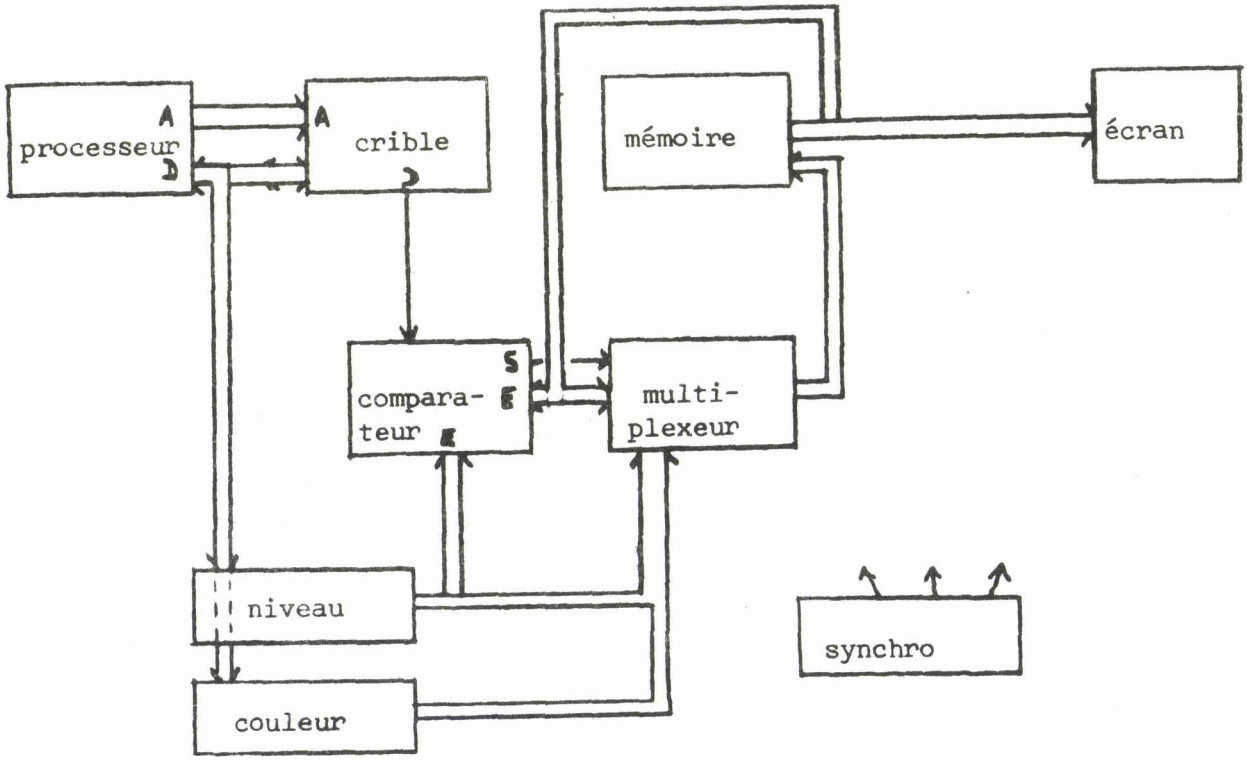
Dans ce cas, lors de la création d'une tache, le système effectuera les opérations suivantes :

- décodage des contours de la tache et remplissage du crible (mémoire à 1 bit par point)

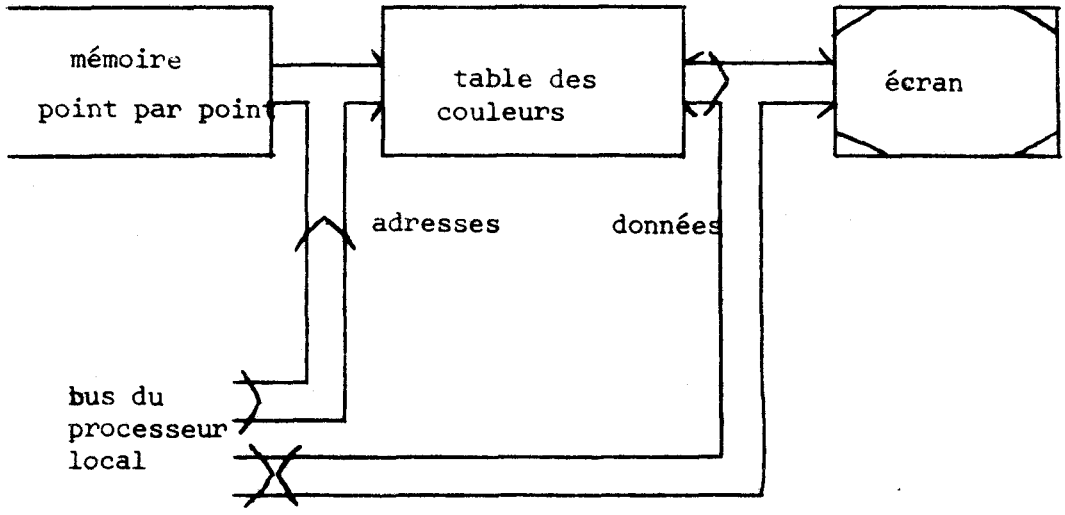
- lecture synchrone par le processeur de niveaux de l'image actuelle et du crible. Dès que l'on rencontre un bord gauche, on compare le niveau dans l'image au niveau de la tache à introduire ; le plus proche indique la nouvelle couleur à visualiser et donc à mémoriser dans la mémoire copie de l'écran/ Ce processus s'arrête à la rencontre d'un bord droit

On met ainsi à jour l'image en une trame par tache à introduire.

Nous obtenons alors l'architecture suivante



Cette architecture est une première étape, qui ne permet que la création de taches. Une deuxième étape consiste à remarquer que niveau et couleur jouent le même rôle, et que l'on peut donc les confondre, en s'imposant par le fait même une seule tache par niveau ; il y a lieu alors de rajouter en sortie de la mémoire point par point une table de fausses couleurs, accessible depuis le processeur local, transformant l'information (niveau - couleur) contenue dans la mémoire point par point en couleur physique.



Une troisième étape consiste à étendre l'utilisation du crible à l'affichage de ligne et non de taches, en rendant programmable le fonctionnement du comparateur de niveaux. Une quatrième consiste à rendre le système bidirectionnel, c'est-à-dire : permettre de

- récupérer tache par tache le contenu vrai de l'image, en précisant simplement les niveaux que l'on cherche à étudier

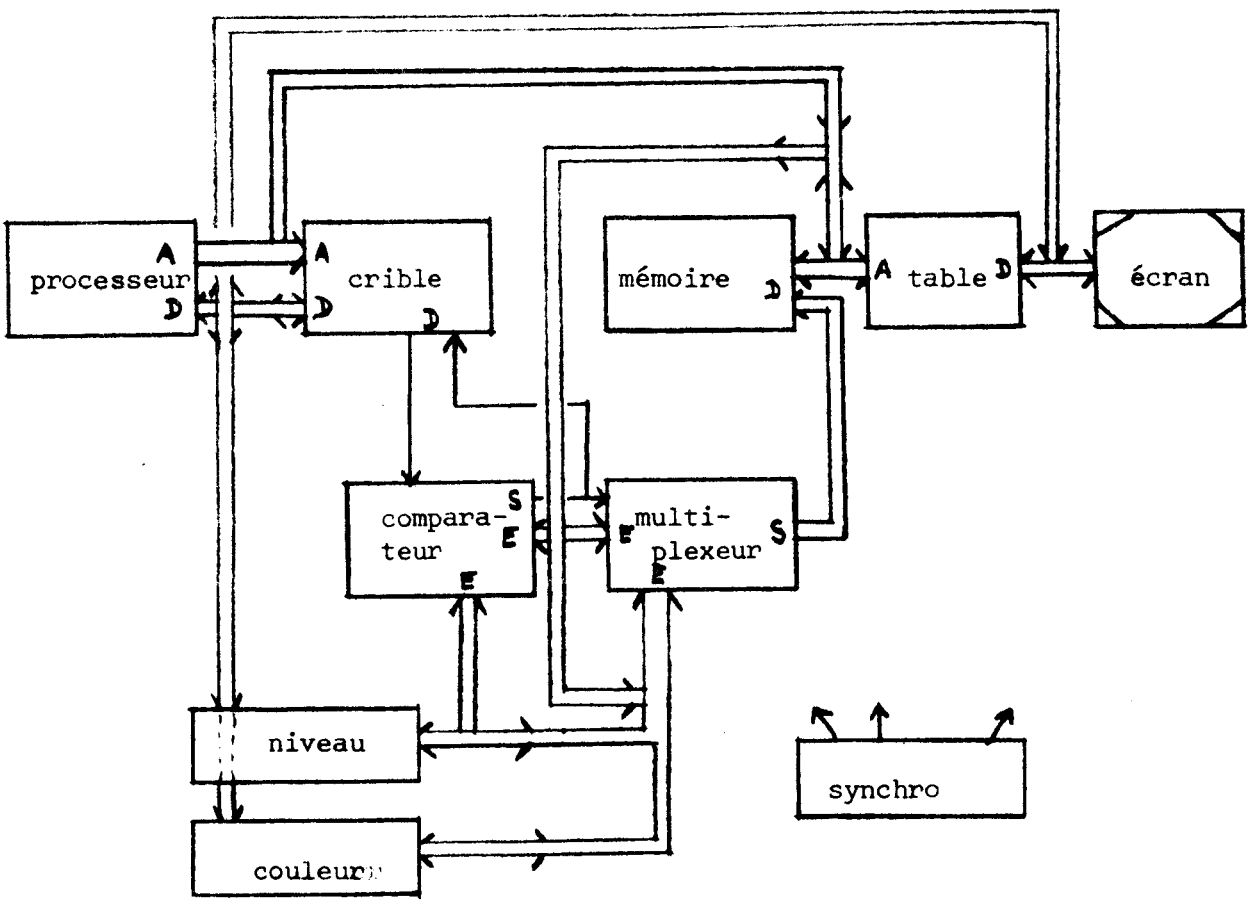
- effacer une tache sur un niveau donné. Bien sûr, il n'est alors pas possible de restituer le fond, à moins de réinscrire les taches qui étaient cachées

- récupérer le niveau (et la couleur) d'un point précisé dans le crible. Ceci permet directement de désigner une tache par un point intérieur visible

Nous pouvons alors effectuer presque toutes les opérations élémentaires souhaitables, à savoir

- créer une ligne ou une tache
- modifier sa couleur
- nommer une tache visible sur l'écran et récupérer sa partie visible
- effacer une tache

L'architecture est alors la suivante



E = entrées

S = sorties

A = adresses

D = données

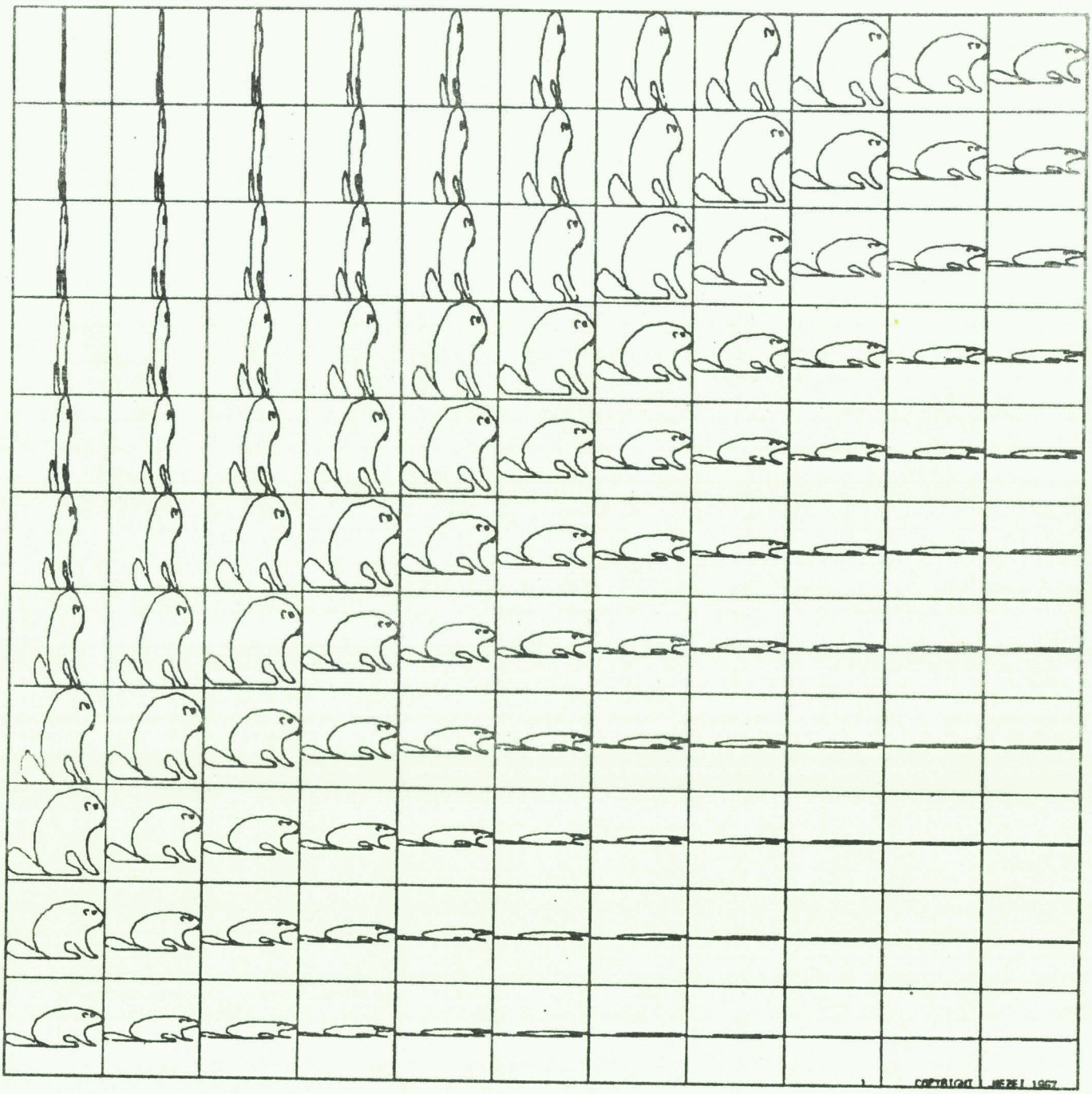
C'est celle-ci qui est effectivement réalisée dans le système STYX et qui est détaillée en Annexe. Notons de plus que cette architecture peut être facilement rendue compatible avec les habituels terminaux à rafraîchissement. De plus la notion de crible permet éventuellement de s'affranchir d'une description par contours et de calculer directement des coordonnées de points.

Il est clair que les performances sont amoindries par rapport au système idéal ; néanmoins seuls sont affectés le temps de réponse (ici 20 ms par tache modifiée) et le coût, dû à la mémoire de rafraîchissement. L'image sous forme de descripteurs se trouvant rejetée au niveau du processeur local, sa taille est quelconque et non limitée, du moins théoriquement.

De plus cette architecture permet certaines manipulations originales de taches, notamment la récupération de la partie visible d'une tache.

4-6 Conclusion

De ces diverses architectures, seule cette dernière nous a paru réalisable. Cependant les plus intéressantes sont sans aucun doute la structure pipe-line/parallèle et surtout la structure à un processeur par tache : il faut là attendre une certaine évolution technologique pour pouvoir songer à réaliser effectivement une telle machine : en particulier, seule une intégration "à la demande" permettrait de réaliser les "processeurs-taches" de façon satisfaisante.



'BEAVER SCALED'

CHAPITRE 5

APPLICATIONS ET VOIES DE RECHERCHE

5 APPLICATIONS ET VOIES DE RECHERCHE

Il importe d'étudier d'une part l'adaptabilité du système à certaines applications existantes, mais aussi les applications potentielles liées à ses caractéristiques spécifiques.

5-1 La notion de tache ([New 71]).

Cette notion, qui inclut la notion usuelle de "trait", demande une étude approfondie des traitements susceptibles d'être effectués. Relativement bien connue dans le domaine de la "graphique", la tache est peu, voire pas, utilisée en informatique graphique ; comment la créer, comment la manipuler, quels avantages en retirer sont autant de questions à reposer de façon générale. Il faut aussi y répondre dans le cadre particulier de ce système, c'est-à-dire élaborer un "logiciel de base" susceptible de

- mettre à la disposition de l'usager un moyen de créer des taches, soit à partir d'une bibliothèque de primitives de formes (cercle, carré, segment, triangle...), soit à l'aide d'une autre description, soit encore à l'aide d'un organe d'entrée adéquat (digitaliseur, caméra, etc...)
- mettre à sa disposition des moyens de manipuler ou de décrire les manipulations désirées sur ces taches ; il paraît utile de disposer de fonctions élémentaires, telles que, par exemple.
  - . mise à l'échelle en abscisse et/ou ordonnée
  - . rotation, translation
  - . inversion, symétrie
  - . désignation
  - . effacement

Cette étude théorique ou liée au système de la notion de tache serait grandement facilitée par l'analyse d'un certain nombre de problèmes où la manipulation de taches paraît a priori apporter quelque chose. De tels problèmes sont par exemple

- la cartographie, c'est-à-dire la représentation sur un fond de carte de données relatives à une région

- l'analyse de données, c'est-à-dire finalement la manipulation, sous forme de matrices, de données diverses (statistiques le plus souvent)

- la création architecturale ou artistique

- le traitement d'images

Cette liste (non exhaustive) n'exclut pas des domaines tels que

- l'aide à l'enseignement
- la création graphique (dont le dessin animé)
- la visualisation de schémas, graphes, formes, résultant de calculs scientifiques.

De ceci devrait découler un noyau commun à la manipulation de taches.

5-2 Caractéristiques spécifiques du système

Le système proposé a des caractéristiques particulières liées au processeur graphique, dont il faut étudier les applications possibles.

5-2-1 Propriétés géométriques ([Skl 76], [Fre 75], [Skl 72], [Fre 77]).

Le mode de description des taches autorise le calcul ou l'évaluation d'un certain nombre de propriétés géométriques, à savoir par exemple

- l'aire d'une tache
- son périmètre
- sa connexité
- son nombre d'Euler

Il facilite également notablement certaines manipulations sur les taches :

- translation (par changement d'origine (s))
- mise à l'échelle d'un facteur 2^k, avec $k \in Z$,

par expansion ou compression du code

- symétrie d'axe vertical, par changement éventuel de l'origine et inversion droite-gauche
- symétrie d'axe horizontal, par inversion du sens de parcours des contours.

5-2-2 Affichage alphanumérique

Si l'on associe à un caractère (alphanumérique, ou quelconque) un code de contours, on peut généraliser l'affichage alphanumérique usuel : l'affichage d'un caractère consistera à préciser divers paramètres

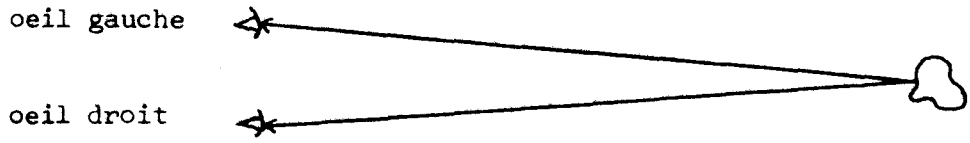
- son origine
- sa couleur et son niveau
- sa taille (facteur de mise à l'échelle)

Chaque caractère est alors un élément de l'image, et non plus un élément "rapporté" artificiellement, comme dans de nombreux systèmes.

5-2-3 Relief ([Rep 72]).

La notion de plans parallèles, donc d'espace à trois dimensions, induit la notion de relief ; la perception de ce relief relève de deux processus distincts :

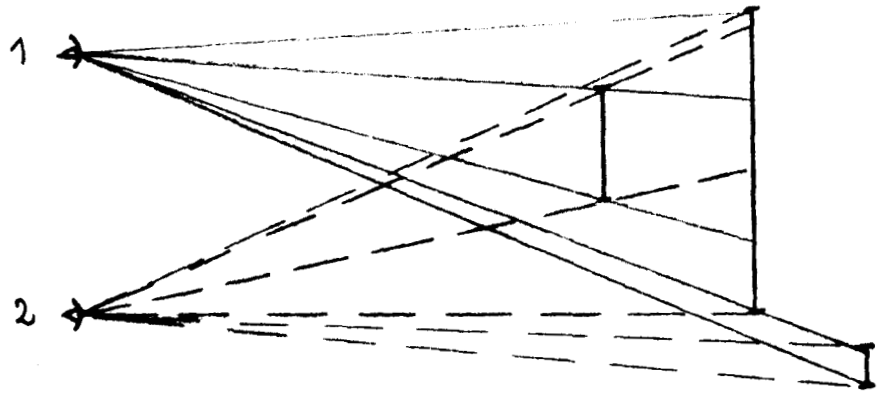
- la vision binoculaire, qui permet d'évaluer la distance de l'observateur à l'objet.



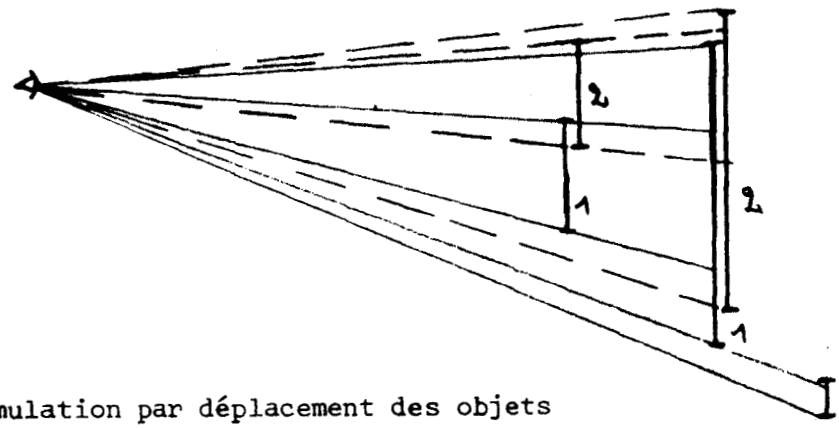
- le déplacement
  - . de l'objet par rapport à l'observateur.
  - ou . de l'observateur par rapport à l'objet.

Ces deux aspects sont identiques si l'on ne considère que l'observateur et l'objet ; il est clair cependant que l'existence d'un décor, c'est-à-dire d'un référentiel les différentie.

Il paraît intéressant ici de s'intéresser au déplacement de l'observateur par rapport à l'objet ; ceci peut être simulé en déplaçant *l'ensemble des objets* de l'image par rapport à l'observateur, mais en tenant compte que le fond ("l'infini" ou "l'horizon") reste immobile, et que c'est le 1er plan qui se déplace le plus.Pour de faibles déplacements, c'est-à-dire pour ceux tels que $\sin\alpha$ peut être "raisonnablement" approximé à α, on doit obtenir une impression de relief, à ceci près que l'observateur reste immobile.

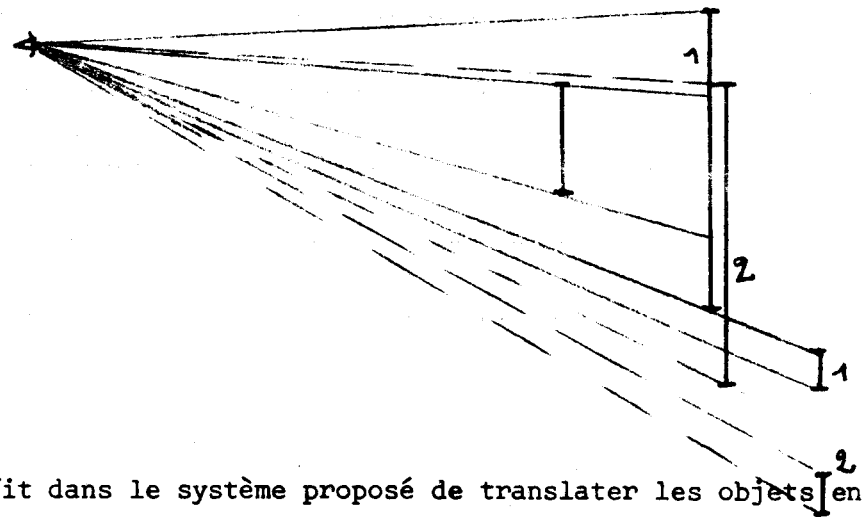


1 et 2 : positions successives de l'observateur



simulation par déplacement des objets immobile

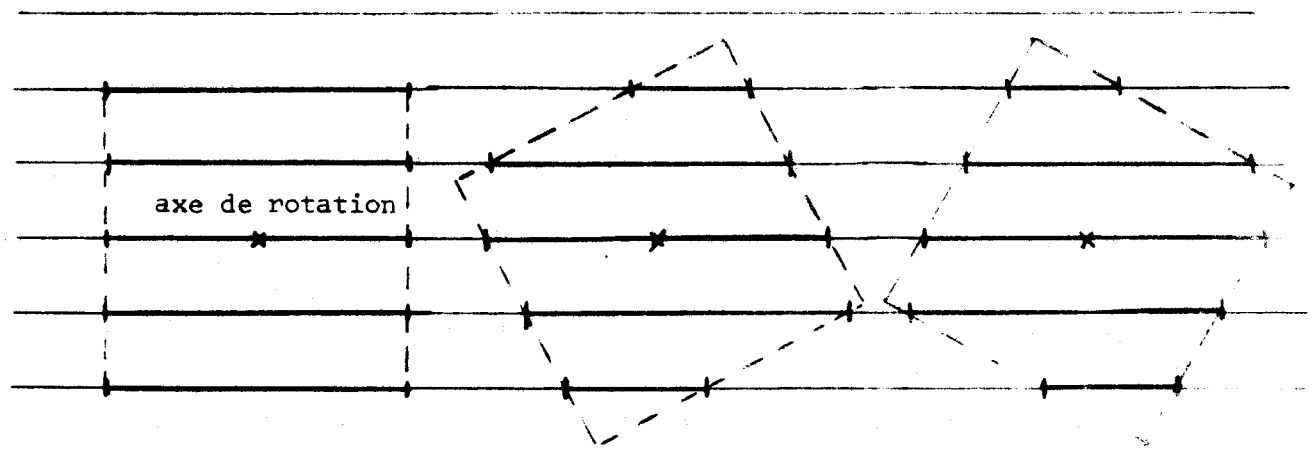
On peut aussi imaginer une solution où c'est le premier plan qui reste immobile (ou encore toute solution intermédiaire).



Il suffit dans le système proposé de translater les objets en fonction de leur position dans l'espace des plans.

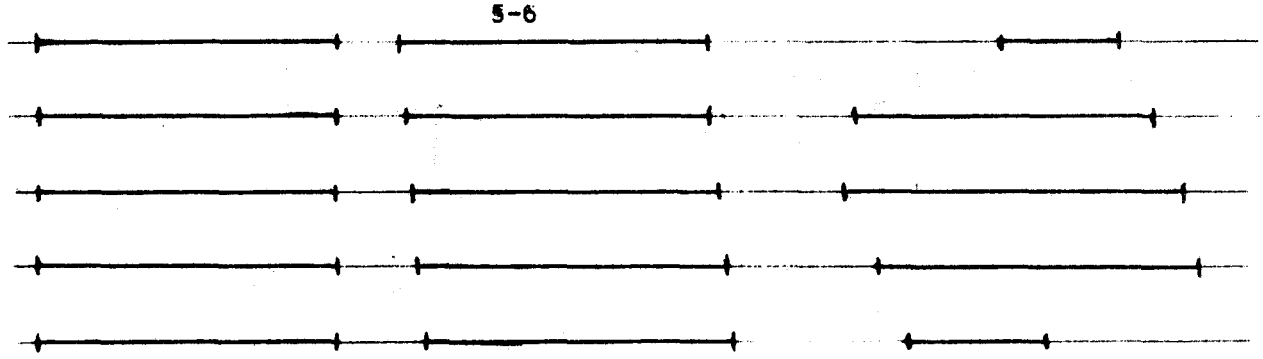
5-2-4 Volumes ([Cro 78]).

L'aspect "relief" évoqué ci-dessus, rapproché avec la description en coupes des volumes proposée, nous incite à imaginer des rotations spatiales de volumes : si nous disposons de vues d'un objet suivant plusieurs angles d'observation, l'approximation précédente peut servir à relier ces vues successives. Ainsi, si l'on dispose de vues d'un cube vertical de 30° en 30°, on aurait

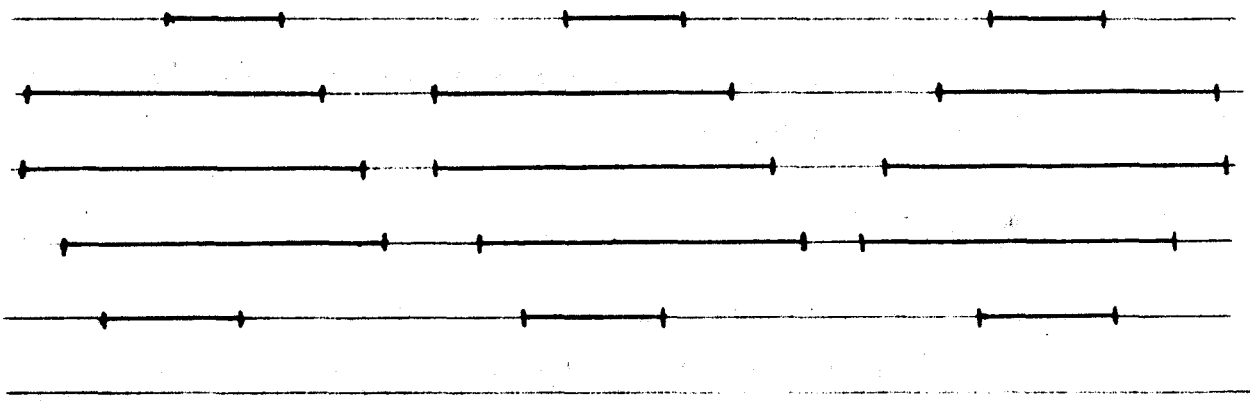


(1) $\alpha = 0°$ (2) $\alpha = 30°$ (3) $\alpha = 60°$

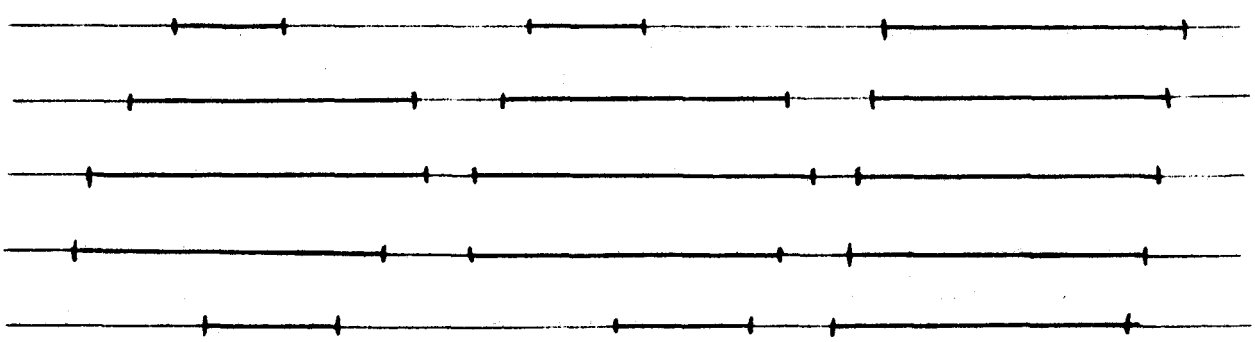
Si on s'autorise une "translation" de $\pm$ 15° par rapport à ces vues réelles, on obtiendrait.



(1) α = 0° (1) + 10° (2) - 10°



(2) α = 30° (2) + 10° (3) - 10°



(3) α = 60° (3) + 10° (1) - 10°

Le codage autorise bien entendu des translations donc des "pseudo-rotations" quelconques.

Ces deux derniers points mériteraient, par leur originalité une attention particulière.

CONCLUSION

L'analyse des systèmes existants, et la mise en évidence de la nécessité de comprimer les informations graphiques nous ont amené à étudier un système graphique basé sur la notion de tache, réalisant un compromis entre

- l'intelligence locale au terminal graphique
- l'intéractivité et la rapidité
- la faisabilité du processeur graphique
- le coût du système

Une réalisation (en partie simulée) a été réalisée qui tient compte de cette étude et de contraintes matérielles diverses ; elle n'est certes pas complétement satisfaisante, mais ouvre plusieurs voies de recherches :

- l'une concerne la notion de tache et son applicabilité dans divers domaines graphiques
- l'autre concerne la réalisation d'un processeur graphique plus conforme aux objectifs initiaux, c'est-à-dire notamment complètement câblé et susceptible d'interpréter directement la notion de tache. Il s'agit en particulier de poursuivre l'étude précise d'un "processeur-tache" susceptible dêtre intégré, qui aurait l'avantage de rendre le terminal extensible quasi-indéfiniment.
- une (ou des) autre (s) concernerai(en)t les problèmes de perception de forme, de relief, de volumes et de mouvement, en liaison avec le mode de description que nous avons décrit.

BIBLIOGRAPHIE

-:-:-:-:-:-:-:-:-:-:-:-

[Agr 75] Agrawala A.K. : On genrating à Line "Parallel" to a Digital Line
ITEC, Oct. 75.

[And 76] Andrews H.C., Patterson C.L. : Outer Product Expansions and their Uses in Digital Image Processing.
ITEC, Vol. C-25, n°2, Feb. 76.

[And 76] Andrews H.C., Patterson C.L. : Digital Interpolation of Discrete Images.
ITEC, Vol. C-25, n° 2, Feb. 76.

[Bar 74] Barrett R.C., Jordan B.W. : Scan Conversion Algorithms for a Cell Organized Rzster Display.
CACM, march 74, Vol. 17, n°3.

[Bas 76] Baskett F., Shuskek L. : The Design of a Low Cost Vidéo Graphics Terminal
Feb. 76 - Stanford - SC - 75-546.

[CIT] Documentation CIT-Alcatel (TRIM)

[Col 76] Colorix : Mémoire de Maîtrise présenté par L. Audoire à l'Université de Paris VIII (76)

[Col 77] Colonna J.F. : Un système multimedia conversationnel
01-Informatique, Oct. 77.

[Com 76] Computer, Vol. 9, n° 8, Aug. 76.

[Com 77] Computer Graphics, Siggraph' 77 proceedings
ACM, Vol. 11, n° 2, Summer 77.

[Gro 78] Crow F.C. : Shaded Computer Graphics in the Entertainment Industry.
Computer, March 78.

[Dac 71] Dacey M.F. : Poly, A two Dimensional Language For a class of Polygons
Pattern Recognition, Vol. 3, 1971,

[Duc 74] Ducrot A. : Principes d'organisation et de réalisation du Système Graphique Interactif GIPSY
THèse de 3ème cycle, Université de Lille, mai 74.

[Duf 73] Duff M.J.B., Watson D.M., Fountain T.J., SHAW G.K. : A callular Logic Array for image Processing
Pattern Recognition, 1973, Vol. 5.

[Fre 61] Freeman H. : On the encoding or arbitrary geometric configurations
ITEC, JUne 61.

[Fre 64] Freeman H., Garder L. : A pictoral jigsaw puzzles
ITEC, April 64.

[Fre 75] Freeman H., Shapira R. : Determining the minimum-area encasing Rectangle for an Arbitrary closed Curve
CACM, Vol. 18, n° 17, July 75.

[Fre 77] Freeman C., Davis L.S. : A Corner-Finding Algorithm for chain-coded Curves
ITEC, March 77.

[Ger 76] Germain-Bonne B., Sablonnière P. : Comparaison des Formes de courbes paramétrées et de leurs approximants - spline
Pub. n° 76, Oct. 76, lab. de calcul de l'Université de Lille.

[Gil 76] Gilbert B.K. et al. : A real-time hardware System for digital processing of wide-band video images.
ITEC, Vol. C-25, n° 11, Nov. 76.

[Gon 77] Gonzales L. : Analyse et réalisation d'un système de traitement d'images couleurs obtenues par télédétection
Mémoire
CNAM (Lille), Déc. 77.

[Gra 71] Gray S.B. : Local Properties of Binary Images in two dimensions
ITEC, Vol. C-20, n° 5, May 71.

[Hab 74] Habili A. : Robinson G.S. : A survey of digital picture coding computer, May 74.

[Hua 75] Huang T.S. : Bounds on the bit rate of linear runlength codes
ITIT, Nov. 75.

[Int] Documentation Intel

[Jor 73] Jordan B.W., Lennon W.J., Holm B.D. : An Improved algorithm for the generation of Nonparametric curves
ITEC, Vol. C-22, n° 12, Déc. 73.

[Jor 73] Jordan B.W., Barrett B.C. : A Scan Conversion Algorithm with reduced storage requirements
CACM, Nov. 73, Vol. 16, n° 11.

[Jor 74] Jordan B.W., Barrett R.C. : A cell organized Raster display for line drawings
CACM, Feb. 74, Vol. 17, n° 2.

[Jud 77] Judice C.N. : Digital video, a buffer-controlled dither processor for animated images
ITEC, Vol. C-25, N° 11, Nov. 77.

[Kre 73] Kreitzer N.H., Fitzgerald W.J. : A video display system for image processing by computer
ITEC, Feb. 73, Vol. C-22, n° 2.

[Kri 62] Krier F. : Les terminaux de visualisation graphique ;
leur raccordement à un Réseau Informatique.
IRISA, Rennes, Nov. 76, Pub. Int. 62.

[Kul 68] Kulsrud H.E. : A general purpose graphic language
CACM., Vol. 11, n° 4, April 68.

[Lim 77] Limb J.O., Rubinstein C.B., Thompson J.E. : Digital Coding of color video signals - A review.
ITEC, Vol. C-25, n° 11, Nov. 77.

[Luc 76] Lucas M. : Les systèmes de visualisation graphique - état des recherches actuelles.
Bulletin de liaison de l'IRIA, n° 25, Avril 76.

[Luc 78] Lucas M. : Contribution à l'étude des techniques de communication graphique avec un ordinateur - éléments de base des logiciels graphiques interactifs
Thèse d'état, Université de Grenoble.

[Mar 76] Martelli A. : An application of Heuristic Search Mathod to edge and contour detection.
CACM, Vol. 19, n° 22, Feb. 76.

[Mat 78] Matic B., Trottier L. : Graphics CRT Displays...
Electronic Design 4, Feb. 78.

[Mer 78] Mériaux M. : STYX, un système de visualisation d'images en couleur par taches.
Congrès AFCET 78, Gif-sur-Yvette.

[Mol 71] Moles A. : Art et Ordinateur - Synthèses contemporaines - Casterman 71.

[Mor 76] Morvan P., Lucas M. : Images et ordinateur
Larousse Université, 76.

[Mye 68] Myer T.H., Sutherland I.E. : On the Design of Display Processors
CACM, Vol. 11, n° 6, June 68.

[Neg 77] Negroponte N. : Raster Scan Approaches to computer graphics
Computer and graphics, Vol. 2, 1977.

[New 71] Newman W.M. : Display procedures
CACM, Oct. 71, Vol. 14, n° 10.

[New 73] Newman W.M., Sproull R. : Principes of interactive computer graphics
Mc Grw Hill. 73.

[New 76] Newman W.M. : Trends in graphic display design
ITEC, Vol. C-25, n° 12, Déc. 76.

[Nol 71] Noll A.M. : Scanned-Display computer graphics
CACM, March 71, Vol. 14, n° 3.

[Pal 76] Palermo F.P., Meder H.G. : Towards a Flexible interactive graphics system
Computer Science, July 16, 1976.

[Per] Documentation SEIN (Pericolor).

[Ray 76] Raymond J., Banerji D.K. : Using a Microprocessor in an Intelligent graphics Terminal
Computer, April 76.

[Rep 72] Repko M.G. : Animated cartoon design with a CRT
The computer journal, Vol. 15, n° 4, 72.

[Roe 77] Roese J.A., Pratt W.K., Robinson G.S. : interframe Cosine Transform Image coding
ITEC, Vol. C-25, n° 11, Nov. 77.

[Sig 76] Sigplan Notices
Vol. 11, n° 6, June 76.

[Ski 72] Sklansky J., Crazin R.L., Hansen B.J. : Minimum perimeter polygons of digitized silhouettes
ITEC, Vol. C-21, n° 3, March. 72.

[Ski 76] Sklansky J., Cordella L.P., Leviadi S. : Parallel Detection of concavities in cellular Blobs.
ITEC, Vol. C-25, n° 2, Feb.76.

[Suz 74] Suzuki Y. and al. : Computer driven color graphic display device with surface painting
compcon 74, 26, 28 Feb. 74.

[Tek] Documentation TEKTRONIX

[Viv 69] Vivier B. : Structure des équipements de visualisation graphique...
L'onde Électrique, Vo. 49, n° 6, Juin 69.

[Who 61] Wholey J.S. : The Coding of pictorial data
ITIT, April 61.

[Wis 77] Wiseman N.E. and al. : On making graphic arts quality output by computer.
The computer journal, Vol. 21, n° 1, 77.

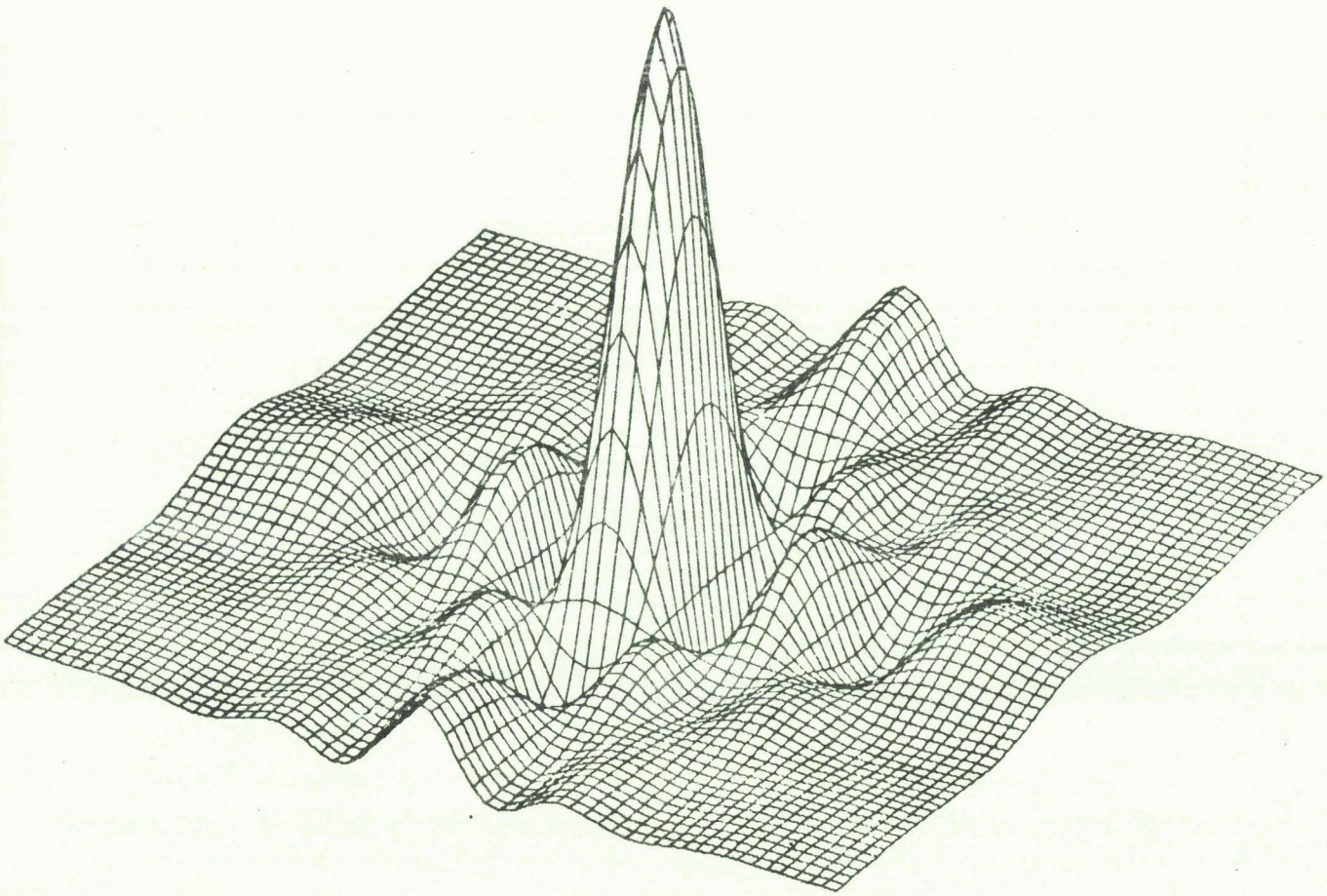
[Yak 76] Yakimovsky Y. : Boundary and object detection in real world images
JACM, Vol. 23, n° 4, Oct. 76.

[Yak 76] Yakimovsky Y., Rayfield M., Eskenazi R. : RAPID, A random access Picture Digitizer, Display, and memory system.
May 76, J.P.L., Pasadena (μ.F) [N 76-23860/9 GA].

[Yal 75] Yalabik N., Cooper D.B. : Compression of contour data through exploiting curve-to-curve dependance
1975, Brown Univ., Providence (μ.F) [N 76-22959/0 GA].

[Zah 72] Zahn C.T., Roskies R.Z. : **Fourier** Descriptors for plane closed curves
ITEC, March 72, Vol. C-21, n° 3.

ITEC : IEEE Transactions on Electronic computers
CACM : Communications of the Association for computing machinery
ITIT : IEEE Transactions on information theory
JACM : Journal of the association for computing Machinery.



ANNEXE

-:-:-:-:-:-:-:-:-

ANNEXE

Le système réalisé

-:-:-:-:-:

Planches a) µ ordinateur INTEL 8080
- architecture du processeur
- jeu d'instruction
- schéma de la carte SBC 80/10
- schéma de la carte SBC 104

b) Convertisseur digital-analogique

c) La mémoire de rafraîchissement
- Notice technique F 16 K-3 Fairchild
- Schéma de la mémoire

d) Le crible
- Notice technique 2114 INTEL
- Schéma du crible

e) Le processeur de mise-à-jour

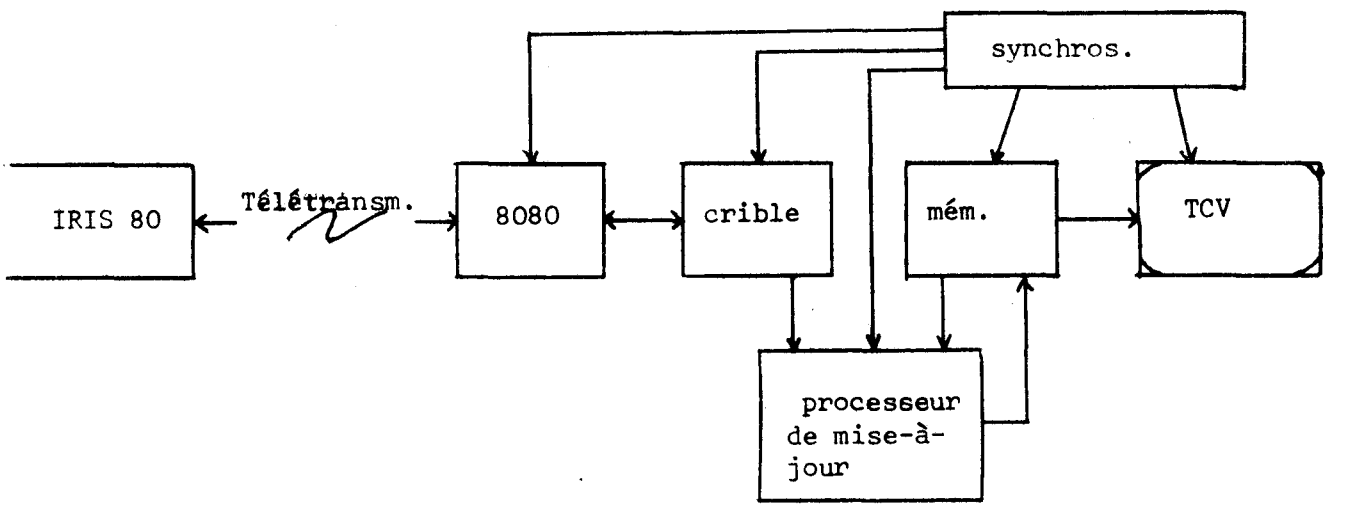
- Schéma

f) Le bloc de synchronisation et de génération de phases

1- Architecture générale

Le dispositif de visualisation est un téléviseur couleur standard. Il est alimenté en permanence par le contenu d'une mémoire de rafraîchissement de 64 K octets. Ce contenu est mis à jour sur l'ordre du processeur local, qui assure également une partie du décodage : le contenu du "crible" est comparé au contenu de la mémoire de rafraîchissement par un processeur de mise à jour, qui décide des données à écrire dans la mémoire de rafraîchissement.

Le processeur local est connectable à l'ordinateur (IRIS 80) du centre de calcul de l'Université. Il s'agit donc d'un système multiprocesseur, que l'on peut représenter de la façon suivante.



2- Le processeur local

Il est chargé

- de réaliser la liaison avec l'IRIS 80
- de réaliser des fonctions simples
- de décoder les contours des objets pour les inscrire dans le crible (simulation d'une partie du processeur graphique)

Il s'agit d'un micro-ordinateur à base de micro-processeur Intel 8080, dans la configuration suivante :

- une carte SBC 80/10 comprenant (entre autre)
  - le processeur
  - 1 K octets de mémoire vive
  - 4 K octets de mémoire morte
  - 48 voies d'entrées-sorties parallèles
  - une voie d'entrée-sortie série

- une extension SBC 104 comprenant
  - 4 K octets de mémoire vive
  - 4 (ou 8) K octets de mémoire morte
  - 48 voies d'entrées-sorties parallèles
  - une voie d'entrée-sortie série
  - un "timer"
  - un registre d'interruptions (8)

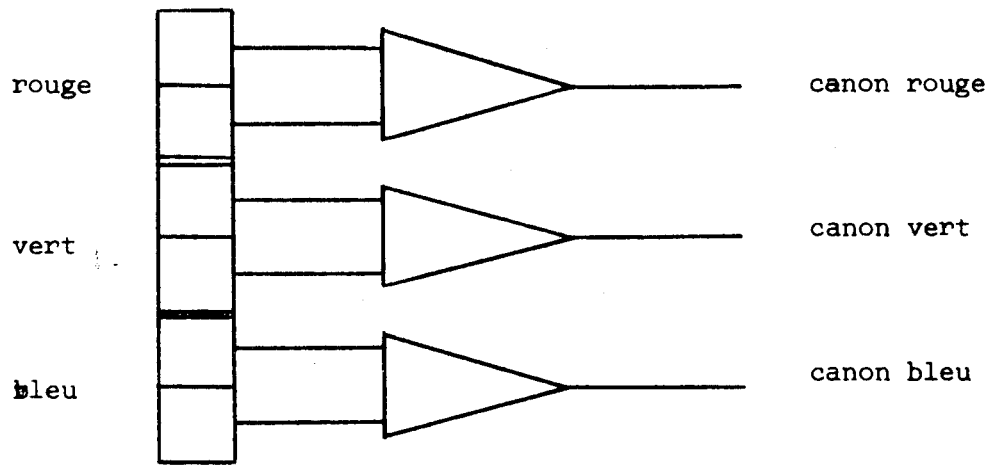
Il est connecté à une télétype, et pourrait l'être à tout autre moyen d'entrée-sortie usuel (écran, imprimante, voyants, etc...) Il est extensible, aussi bien en processeurs qu'en mémoire, entrées-sorties, on interfaces particulières (pour disques souples en particulier). La liaison MODEM vers l'IRIS 80 peut se faire de 110 à 9600 bauds en synchrone ou asynchrone (sont utilisés ici 1200 Bds asychrone, ou 4800 Bds synchrone).

L'interface avec le crible se fait à l'aide du bus externe et de ports d'entrées-sorties parallèles.

3- L'écran

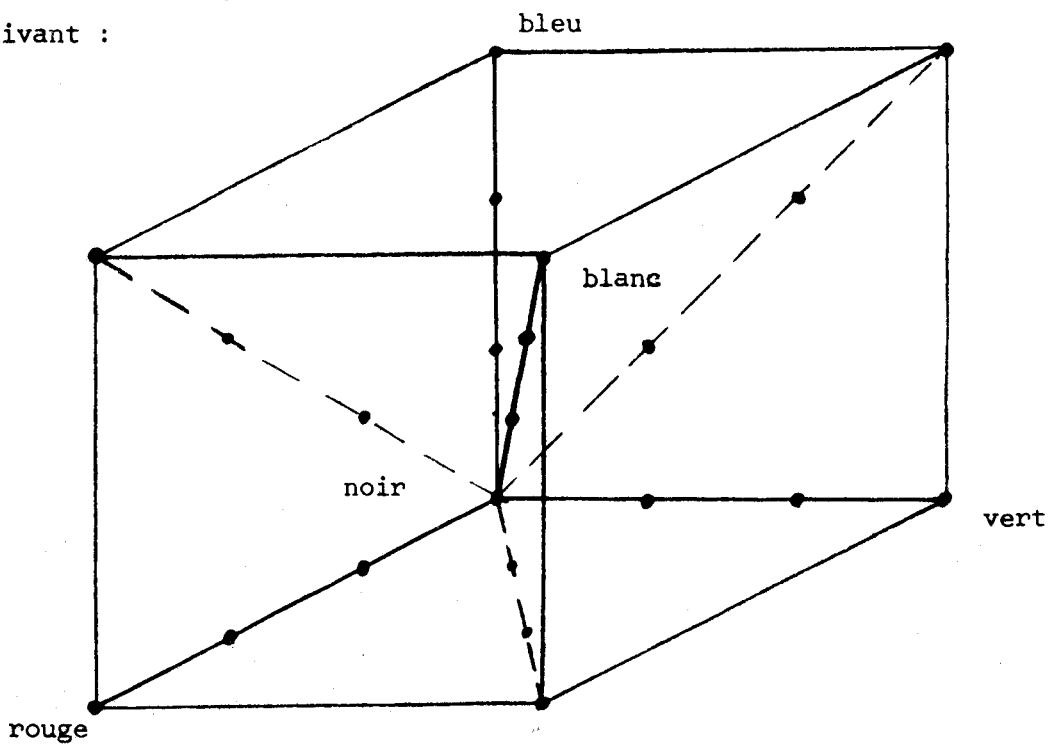
Comme nous l'avons indiqué, il s'agit d'un téléviseur standard couleur (modèle de base de la gamme ITT), équipé d'un tube P.I.L à 90°, dont la qualité est satisfaisante. Une étude du masque et des circuits annexes a montré qu'il était raisonnable de se limiter à un affichage de 256 lignes de 256 points, par deux 1/2 trames identiques ; chaque point dure 125 ns et occupe environ 1 mm × mm. Pour ce faire, il faut néanmoins attaquer directement le tube, sans passer par un modulateur (SECAM), qui limiterait la bande passante d'une façon inacceptable.

On a réalisé une interface digital-analogique convertissant les informations digitales provenant du système en 3 niveaux de tension appliqués aux amplificateurs commandant les 3 canons (rouge, vert, et bleu) du tube. Pour des raisons pratiques, nous nous sommes limités à 2 bits par couleur fondamentale, mais ceci peut être étendu sans difficulté s'il en était besoin sans remettre le reste du système en cause.



convertisseurs D/A

Nous disposons donc de 64 teintes, réparties dans le cube de couleurs suivant :



(Nota : ceci pourra paraître trop, ou trop peu, suivant l'utilisation !)

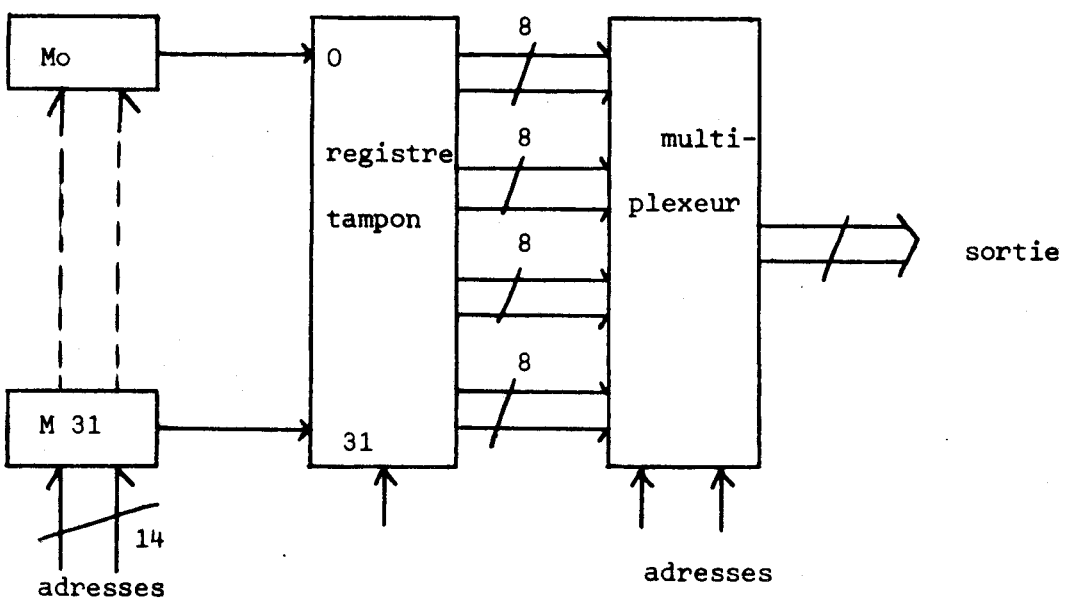
On a également réalisé un bloc de synchronisation fournissant :
- les signaux de synchronisation trame : 50 Hz (soit 20 ms)
- les signaux de synchronisation ligne : 15 625 Hz (64 µs)
- les signaux de commande de la mémoire de rafraîchissement :
  - début et fin de trame affichée
  - début et fin de ligne affichée
  - horloge (période = 125 ns = durée d'un point)
  - adresse (du point à visualiser) (dans la mémoire de rafraîchissement).

4- La mémoire de rafraîchissement

Elle a été réalisée à l'aide de 32 boîtiers de mémoires dynamiques (F 16 K-3 Fairchild). En effet, l'aspect périodique des accès entraîne la disparition du problème du rafraîchissement de ces mémoires, sous réserve de bien choisir le séquencement des adresses.

Ces mémoires ne pouvant fonctionner à 8 MHz, nous avons choisi de travailler sur 4 points en parallèle.

A ce niveau, un point est codé sur un octet ; les 32 boîtiers fonctionnent donc en parallèle avec un temps de cycle de 500 ns ; les sorties sont mémorisées et multiplexées



En sortie, on a disposé une table de transcodage, constituée d'une mémoire vive accessible de l'extérieur (du 8080) en lecture-écriture ; elle est adressée par l'octet sortant du multiplexeur et fournit 6 bits (3 × 2) destinés aux trois convertisseurs analogique-numérique.

Les mémoires dynamiques alimentent également, en phase de mise-à-jour, le processeur correspondant, qui en retour fournit les données à écrire ; les opérations de lecture et d'écriture sont réalisées simultanément.

Dans cette configuration, la mémoire de rafraîchissement est inaccessible directement de l'extérieur. Cet accès ne pourrait se faire qu'en prévoyant un rafraîchissement annexe des mémoires.

Les mémoires dynamiques ont posé divers problèmes annexes, mais assez difficiles à résoudre : ce sont

- le multiplexage des adresses : l'adressage se fait en deux fois 7 bits, c'est-à-dire en lignes et colonnes
- la génération des "strobes" d'adresses et des signaux de lecture-écriture.
- la nécessité ici de trois alimentations, soigneusement découplées (car les 32 boîtiers commutent simultanément, à une fréquence de 8 MHz).

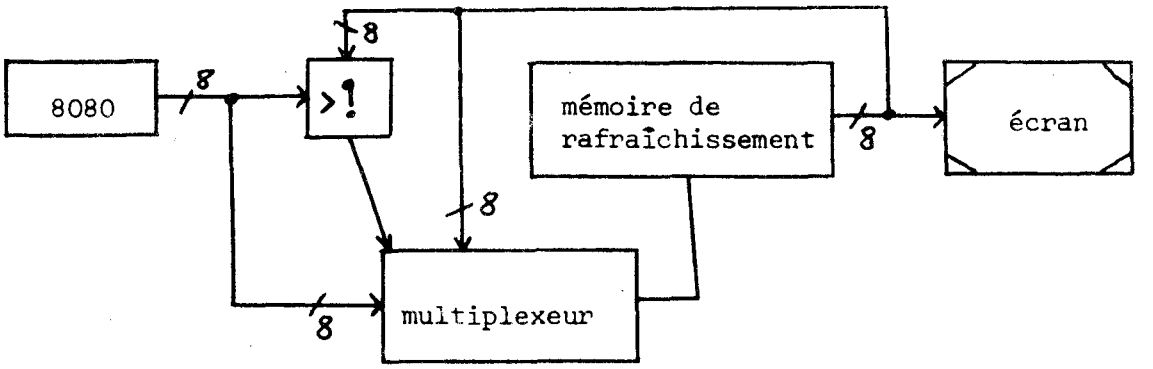
5- Le crible et le processeur de mise-à-jour

Le crible lui-même contient le contour de la tache à introduire. Il s'agit donc d'une mémoire à un bit par point d'image ; comme la mise à jour s'effectue en synchronisme avec la mémoire de rafraîchissement, on a aussi regroupé les points en groupe de 4. Il a été réalisé à l'aide de 16 mémoires statiques de 1 K mots de 4 bits (2114 Intel)

Il est accessible

- soit aléatoirement par le processeur local (8080)
- soit systèmatiquement, avec les mêmes adresses que la mémoire de rafraîchissement, pendant une phase de mise à jour

Le niveau de la tache concernée est fourni par le 8080 (sur un port parallèle) au processeur de mise à jour. Celui-ci compare, s'il y a lieu ce niveau au niveau contenu dans la mémoire de rafraîchissement, et détermine lequel doit y être réécrit.



6- Phases de fonctionnement

On peut distinguer plusieurs phases de fonctionnement

phase 1 (normale) : le 8080 peut accéder au crible (en lecture/écriture), qui est considéré comme appartenant à son espace mémoire. Une fois le remplissage effectué, et le niveau présenté sur le port parallèle, le 8080 envoie au processeur mise à jour un signal de "strobe".

phase 2 (mise à jour) : au vu de ce signal, et en synchronisme avec le balayage T.V., les opérations de comparaison et de réécriture s'effectue, et durent une demie-trame (20 ms). Au terme de cette phase, on démarre la

phase 2 bis (r-a-z) : qui remet à zéro le crible (durée 20 ms). Alors le processeur MAJ renvoie un accusé de réception au 8080, qui retourne en phase 1.

phase 3 : le 8080 a accès à n'importe quel moment à la table de transcodage, qu'il considère comme faisant partie de son espace mémoire.

Durant les phase 2 et 2 bis, l'accès au crible est interdit (exclusion mutuelle) ; la phase 2 bis évite un balayage fastidieux par le 8080 pour remettre à 0 le crible.

Ces phases sont matérialisées par un circuit d'horloges, qui génère les signaux nécessaires aux contrôles des échanges à l'aide de ceux provenant du 8080 et du bloc de synchronisation.

7- Extensions

On peut envisager des extensions

- des fonctions du crible et du processeur local : fonctionnement bidirectionnel, effacement de tache, visualisation d'un seul plan,...
- du processeur local : mémoire, logiciel, mémoires de masse, périphériques usuels.
- du système : adjonction de moyens de communication graphique : photostyle, marqueur, boule roulante, dispositif de sortie sur papier (appareil photo, imprimante), clavier de fonctions spécifiques...

Ne sont envisagés pour le moment que l'adjonction de nouvelles fonctions au crible, d'une mémoire de masse au 8080 (disque souple), d'un photostyle.

Silicon Gate MOS 8080

SINGLE CHIP 8-BIT N-CHANNEL MICROPROCESSOR

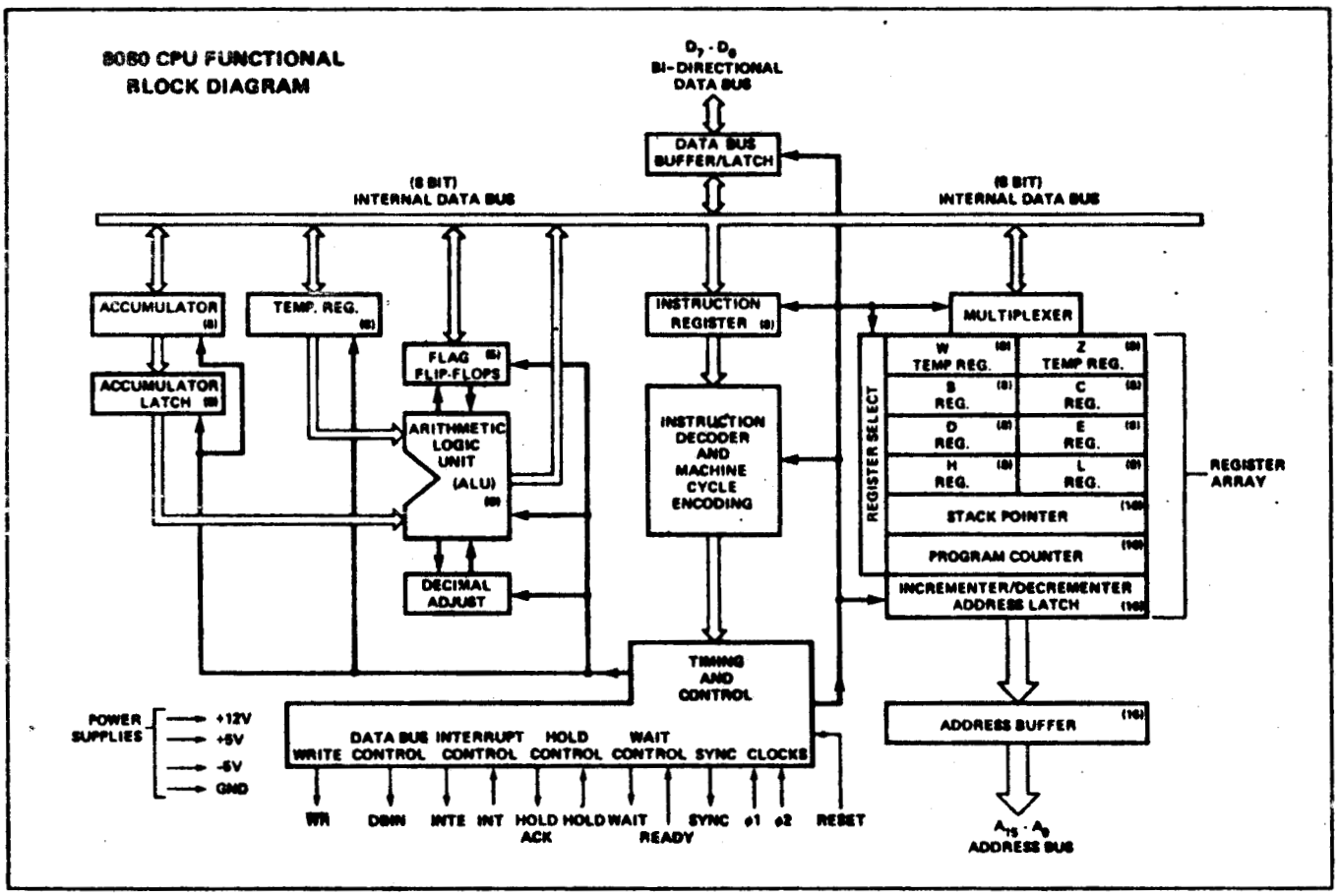
- 2 μs Instruction Cycle
- Powerful Problem Solving Instruction Set
- Six General Purpose Registers and an Accumulator
- Sixteen Bit Program Counter for Directly Addressing up to 64K Bytes of Memory
- Sixteen Bit Stack Pointer and Stack Manipulation Instructions for Rapid Switching of the Program Environment
- Decimal, Binary and Double Precision Arithmetic
- Ability to Provide Priority Vectored Interrupts
- 512 Directly Addressed I/O Ports

The Intel 8080 is a complete 8-bit parallel central processing unit (CPU). It is fabricated on a single LSI chip using Intel's n-channel silicon gate MOS process. This offers the user a high performance solution to control and processing applications.

The 8080 contains six 8-bit general purpose working registers and an accumulator. The six general purpose registers may be addressed individually or in pairs providing both single and double precision operators. Arithmetic and logical instructions set or reset four testable flags. A fifth flag provides decimal arithmetic operation.

The 8080 has an external stack feature wherein any portion of memory may be used as a last in/first out stack to store/retrieve the contents of the accumulator, flags, program counter and all of the six general purpose registers. The sixteen bit stack pointer controls the addressing of this external stack. This stack gives the 8080 the ability to easily handle multiple level priority interrupts by rapidly storing and restoring processor status. It also provides almost unlimited subroutine nesting.

This microcoprocessor has been designed to simplify systems design. Separate 16-line address and 8-line bidirectional data busses are used to facilitate easy interface to memory and I/O. Signals to control the interface to memory and I/O are provided directly by the 8080. Ultimate control of the address and data busses resides with the HOLD signal. It provides the ability to suspend processor operation and force the address and data busses into a high impedance state. This permits OR-tying these busses with other controlling devices for (DMA) direct memory access or multi-processor operation.

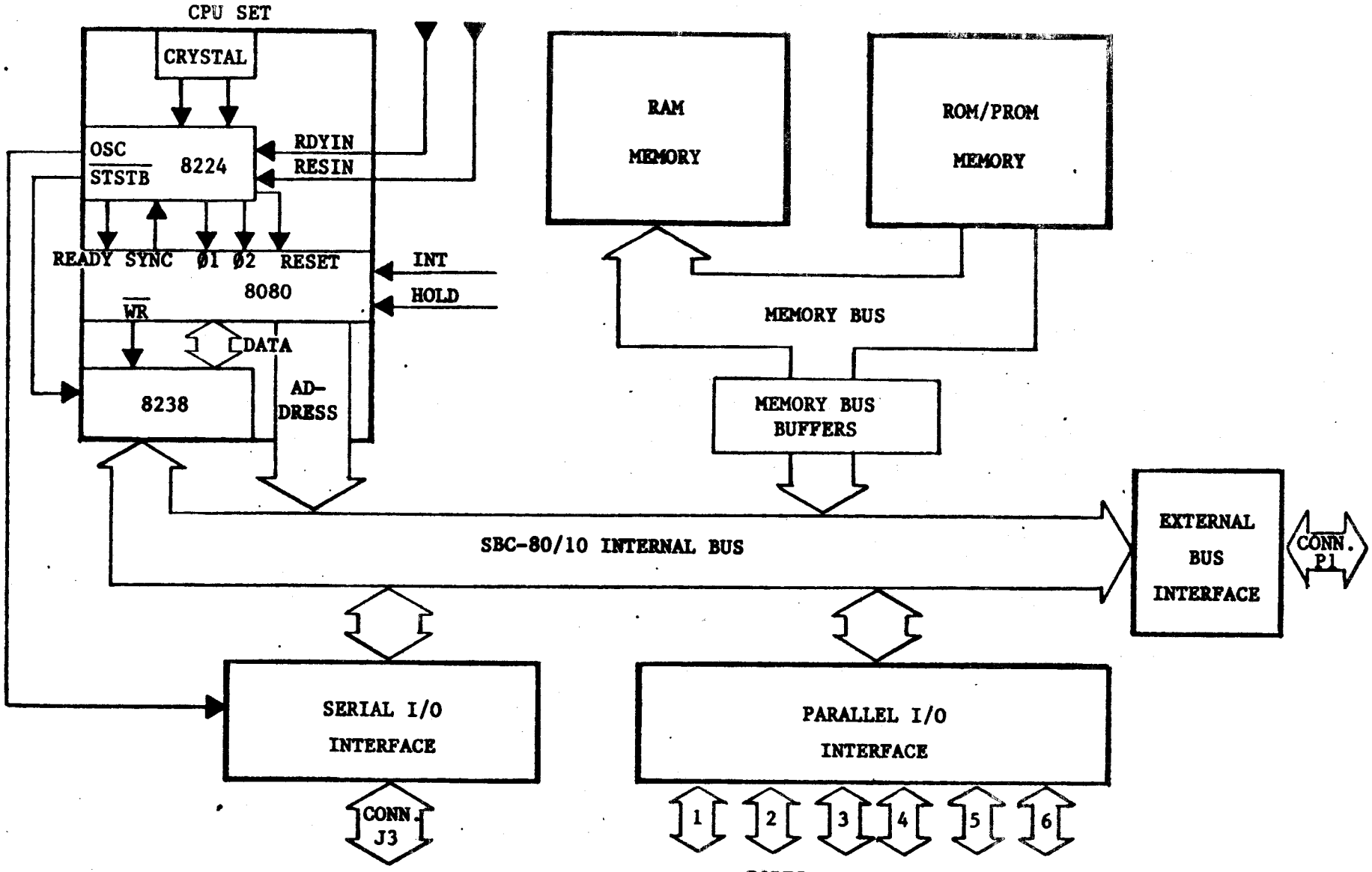


INSTRUCTION SET

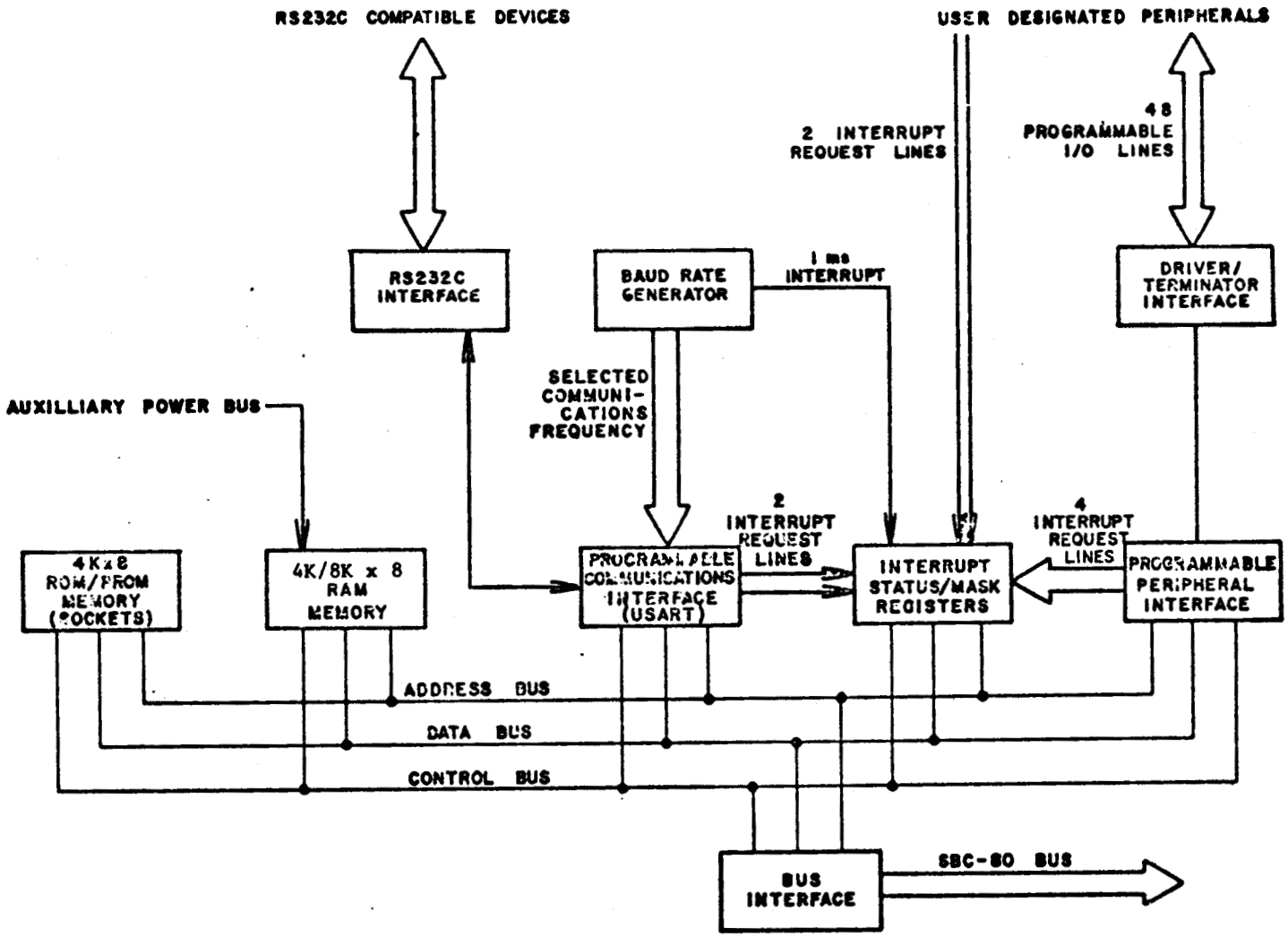
Summary of Processor Instructions

MNEMONIC	DESCRIPTION	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	CLOCK[2] CYCLES
$MOV_{r1,r2}$	Move register to register	0	1	D	D	D	S	S	S	5
MOV M,r	Move register to memory	0	1	1	1	0	S	S	S	7
MOV r,M	Move memory to register	0	1	D	D	D	1	1	0	7
HLT	Halt	0	1	1	1	0	1	1	0	7
MVI r	Move immediate register	0	0	D	D	D	1	1	0	7
MVI M	Move immediate memory	0	0	1	1	0	1	1	0	10
INR r	Increment register	0	0	D	D	D	1	0	0	5
DCR r	Decrement register	0	0	D	D	D	1	0	1	5
INR M	Increment memory	0	0	1	1	0	1	0	0	10
DCR M	Decrement memory	0	0	1	1	0	1	0	1	10
ADD r	Add register to A	1	0	0	0	0	S	S	S	4
ADC r	Add register to A with carry	1	0	0	0	1	S	S	S	4
SUB r	Subtract register from A	1	0	0	1	0	S	S	S	4
SBB r	Subtract register from A with borrow	1	0	0	1	1	S	S	S	4
ANA r	And register with A	1	0	1	0	0	S	S	S	4
XRA r	Exclusive Or register with A	1	0	1	0	1	S	S	S	4
ORA r	Or register with A	1	0	1	1	0	S	S	S	4
CMP r	Compare register with A	1	0	1	1	1	S	S	S	4
ADD M	Add memory to A	1	0	0	0	0	1	1	0	7
ADC M	Add memory to A with carry	1	0	0	0	1	1	1	0	7
SUB M	Subtract memory from A	1	0	0	1	0	1	1	0	7
SBB M	Subtract memory from A with borrow	1	0	0	1	1	1	1	0	7
ANA M	And memory with A	1	0	1	0	0	1	1	0	7
XRA M	Exclusive Or memory with A	1	0	1	0	1	1	1	0	7
ORA M	Or memory with A	1	0	1	1	0	1	1	0	7
CMP M	Compare memory with A	1	0	1	1	1	1	1	0	7
ADI	Add immediate to A	1	1	0	0	0	1	1	0	7
ACI	Add immediate to A with carry	1	1	0	0	1	1	1	0	7
SUI	Subtract immediate from A	1	1	0	1	0	1	1	0	7
SBI	Subtract immediate from A with borrow	1	1	0	1	1	1	1	0	7
ANI	And immediate with A	1	1	1	0	0	1	1	0	7
XRI	Exclusive Or immediate with A	1	1	1	0	1	1	1	0	7
ORI	Or immediate with A	1	1	1	1	0	1	1	0	7
CPI	Compare immediate with A	1	1	1	1	1	1	1	0	7
RLC	Rotate A left	0	0	0	0	0	1	1	1	4
RRC	Rotate A right	0	0	0	0	1	1	1	1	4
RAL	Rotate A left through carry	0	0	0	1	0	1	1	1	4
RAR	Rotate A right through carry	0	0	0	1	1	1	1	1	4
JMP	Jump unconditional	1	1	0	0	0	0	1	1	10
JC	Jump on carry	1	1	0	1	1	0	1	0	10
JNC	Jump on no carry	1	1	0	1	0	0	1	0	10
JZ	Jump on zero	1	1	0	0	1	0	1	0	10
JNZ	Jump on no zero	1	1	0	0	0	0	1	0	10
JP	Jump on positive	1	1	1	1	0	0	1	0	10
JM	Jump on minus	1	1	1	1	1	0	1	0	10
JPE	Jump on parity even	1	1	1	0	1	0	1	0	10
JPO	Jump on parity odd	1	1	1	0	0	0	1	0	10
CALL	Call unconditional	1	1	0	0	1	1	0	1	17
CC	Call on carry	1	1	0	1	1	1	0	0	11/17
CNC	Call on no carry	1	1	0	1	0	1	0	0	11/17
CZ	Call on zero	1	1	0	0	1	1	0	0	11/17
CNZ	Call on no zero	1	1	0	0	0	1	0	0	11/17
CP	Call on positive	1	1	1	1	0	1	0	0	11/17
CM	Call on minus	1	1	1	1	1	1	0	0	11/17
CPE	Call on parity even	1	1	1	0	1	1	0	0	11/17
CPO	Call on parity odd	1	1	1	0	0	1	0	0	11/17
RET	Return	1	1	0	0	1	0	0	1	10
RC	Return on carry	1	1	0	1	1	0	0	0	5/11
RNC	Return on no carry	1	1	0	1	0	0	0	0	5/11
RZ	Return on zero	1	1	0	0	1	0	0	0	5/11
RNZ	Return on no zero	1	1	0	0	0	0	0	0	5/11
RP	Return on positive	1	1	1	1	0	0	0	0	5/11
RM	Return on minus	1	1	1	1	1	0	0	0	5/11
RPE	Return on parity even	1	1	1	0	1	0	0	0	5/11
RPO	Return on parity odd	1	1	1	0	0	0	0	0	5/11
RST	Restart	1	1	A	A	A	1	1	1	11
IN	Input	1	1	0	1	1	0	1	1	10
OUT	Output	1	1	0	1	0	0	1	1	10
LXI B	Load immediate register Pair B & C	0	0	0	0	0	0	0	1	10
LXI D	Load immediate register Pair D & E	0	0	0	1	0	0	0	1	10
LXI H	Load immediate register Pair H & L	0	0	1	0	0	0	0	1	10
LXI SP	Load immediate stack pointer	0	0	1	1	0	0	0	1	10
PUSH B	Push register Pair B & C on stack	1	1	0	0	0	1	0	1	11
PUSH D	Push register Pair D & E on stack	1	1	0	1	0	1	0	1	11
PUSH H	Push register Pair H & L on stack	1	1	1	0	0	1	0	1	11
PUSH PSW	Push A and Flags on stack	1	1	1	1	0	1	0	1	11
POP B	Pop register pair B & C off stack	1	1	0	0	0	0	0	1	10
POP D	Pop register pair D & E off stack	1	1	0	1	0	0	0	1	10
POP H	Pop register pair H & L off stack	1	1	1	0	0	0	0	1	10
POP PSW	Pop A and Flags off stack	1	1	1	1	0	0	0	1	10
STA	Store A direct	0	0	1	1	0	0	1	0	13
LDA	Load A direct	0	0	1	1	1	0	1	0	13
XCHG	Exchange D & E, H & L Registers	1	1	1	0	1	0	1	1	4
XTHL	Exchange top of stack H & L	1	1	1	0	0	0	1	1	18
SPHL	H & L to stack pointer	1	1	1	1	1	0	0	1	5
PCHL	H & L to program counter	1	1	1	0	1	0	0	1	5
DAD B	Add B & C to H & L	0	0	0	0	1	0	0	1	10
DAD D	Add D & E to H & L	0	0	0	1	1	0	0	1	10
DAD H	Add H & L to H & L	0	0	1	0	1	0	0	1	10
DAD SP	Add stack pointer to H & L	0	0	1	1	1	0	0	1	10
STAX B	Store A indirect	0	0	0	0	0	0	1	0	7
STAX D	Store A indirect	0	0	0	1	0	0	1	0	7
LDAX B	Load A indirect	0	0	0	0	1	0	1	0	7
LDAX D	Load A indirect	0	0	0	1	1	0	1	0	7
INX B	Increment B & C registers	0	0	0	0	0	0	1	1	5
INX D	Increment D & E registers	0	0	0	1	0	0	1	1	5
INX H	Increment H & L registers	0	0	1	0	0	0	1	1	5
INX SP	Increment stack pointer	0	0	1	1	0	0	1	1	5
DCX B	Decrement B & C	0	0	0	0	1	0	1	1	5
DCX D	Decrement D & E	0	0	0	1	1	0	1	1	5
DCX H	Decrement H & L	0	0	1	0	1	0	1	1	5
DCX SP	Decrement stack pointer	0	0	1	1	1	0	1	1	5
CMA	Complement A	0	0	1	0	1	1	1	1	4
STC	Set carry	0	0	1	1	0	1	1	1	4
CMC	Complement carry	0	0	1	1	1	1	1	1	4
DAA	Decimal adjust A	0	0	1	0	0	1	1	1	4
SHLD	Store H & L direct	0	0	1	0	0	0	1	0	16
LHLD	Load H & L direct	0	0	1	0	1	0	1	0	16
EI	Enable Interrupts	1	1	1	1	1	0	1	1	4
DI	Disable Interrupt	1	1	1	1	0	0	1	1	4
NOP	No-operation	0	0	0	0	0	0	0	0	4

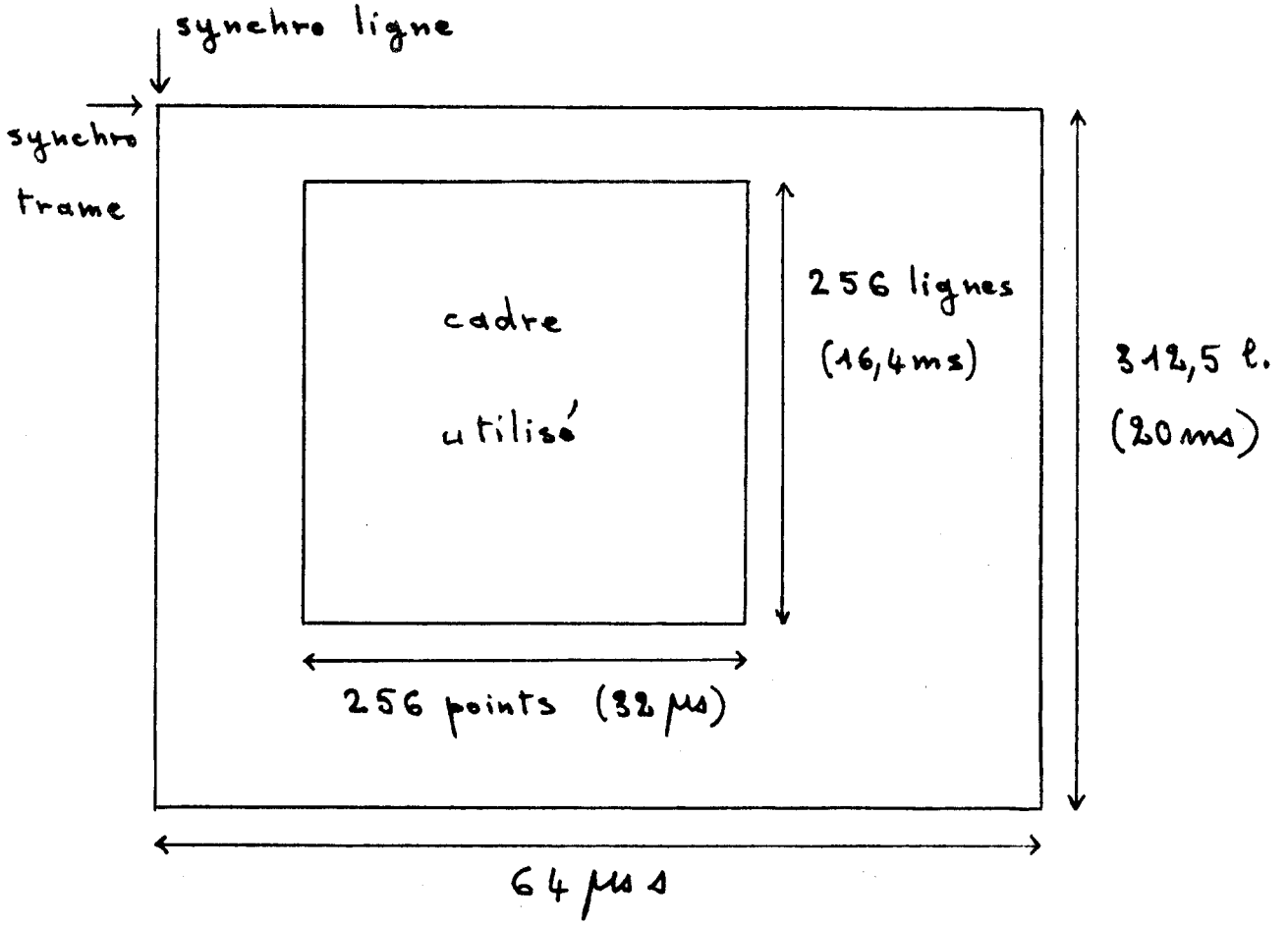
NOTES: 1. DDD or SSS - 000 B - 001 C - 010 D - 011 E - 100 H - 101 L - 110 Memory - 111 A.
2. Two possible cycle times, (5/11) indicate instruction cycles dependent on condition flags.



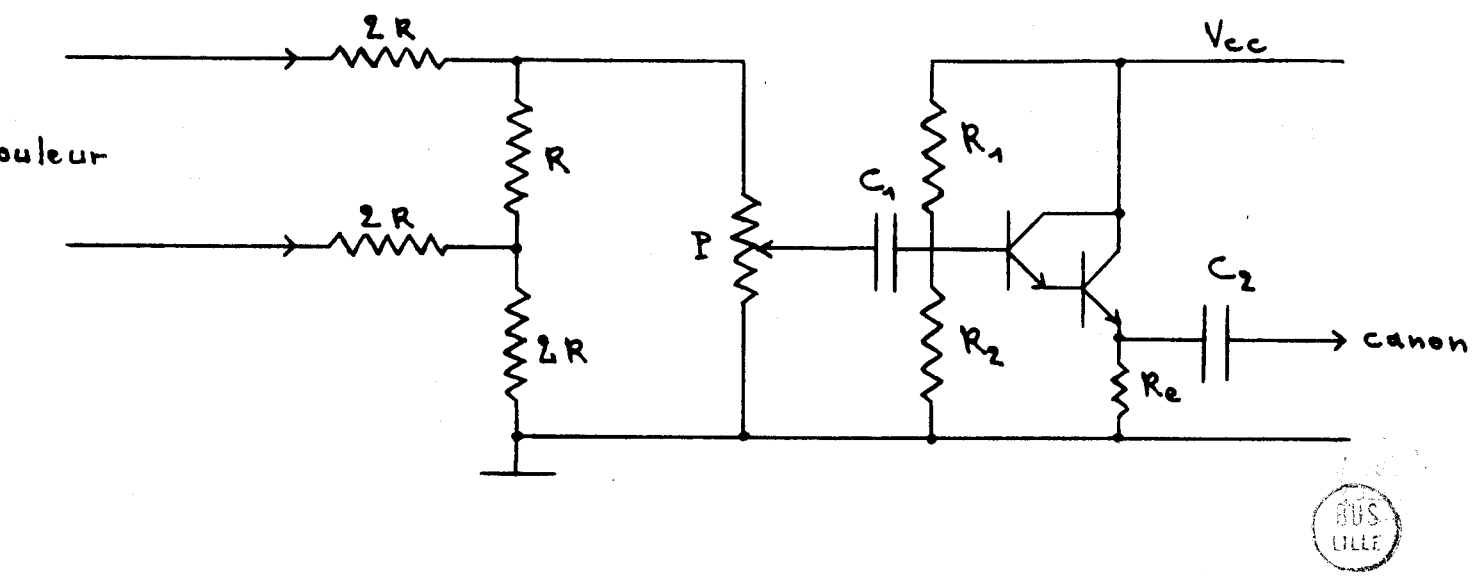
SBC-80/10 FUNCTIONAL BLOCK DIAGRAM



SBC 104/108 FUNCTIONAL BLOCK DIAGRAM



L'ECRAN



L'INTERFACE TV

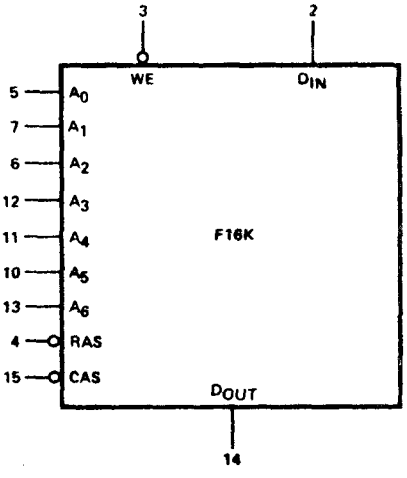
16,384 BIT DYNAMIC RANDOM ACCESS MEMORY

FAIRCHILD ISOPLANAR SILICON GATE MOS

GENERAL DESCRIPTION – The F16K is a 16,384-bit MOS Dynamic Random Access Memory configured as 16,384 one-bit words, and which is manufactured using Fairchild's n-channel silicon gate, double-poly Isoplanar process. The use of the single-transistor memory cell along with address multiplexing techniques permits the packaging of the F16K in a standard 16-pin dual in-line package. The use of this package allows construction of highly dense memory systems utilizing widely available automated testing and insertion equipment. Furthermore, the pinout, timing and organizational characteristics of the F16K are essentially the same as Fairchild's 4K dynamic RAMs, the M4027 and the 4096. Thus the F16K is ideally suited for easy upgrading of M4027 or 4096-based systems.

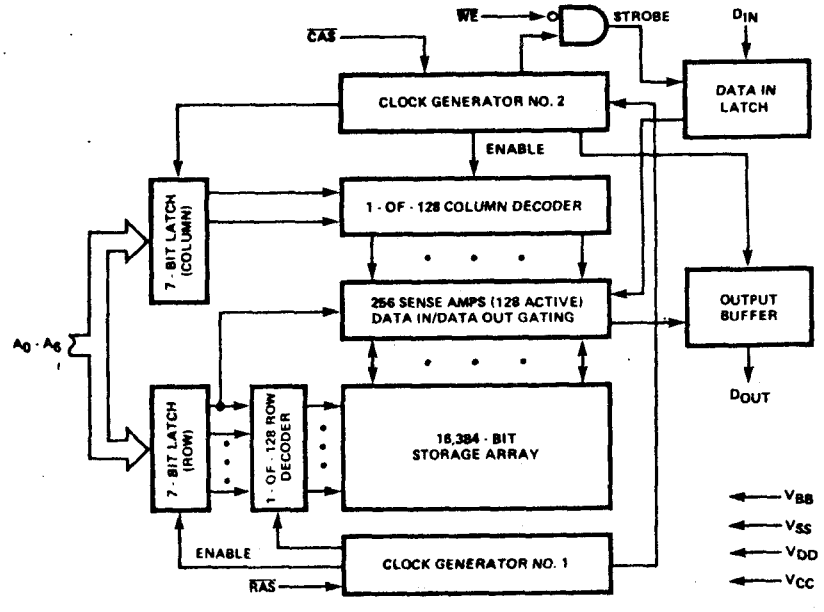
- INDUSTRY STANDARD 16-PIN DUAL IN-LINE PACKAGE
- LOW CAPACITANCE, TTL-COMPATIBLE INPUTS (INCLUDING CLOCKS)
- ON-CHIP ADDRESS AND INPUT DATA LATCHES
- 3-STATE TTL-COMPATIBLE OUTPUT WITH DATA VALID TIME CONTROLLED BY $\overline{CAS}$
- COMMON I/O CAPABILITY
- TWO DIMENSIONAL SELECTION BY DECODING BOTH $\overline{RAS}$ AND $\overline{CAS}$
- STANDARD 10% SUPPLIES (+12 V, +5 V, and −5 V)
- FLEXIBLE TIMING WITH PAGE-MODE AND EXTENDED PAGE BOUNDARIES
- 128-CYCLE $\overline{RAS}$-ONLY REFRESH
- PINOUT COMPATIBLE WITH M4027 OR 4096 FOR EASY SYSTEM UPGRADING

LOGIC SYMBOL

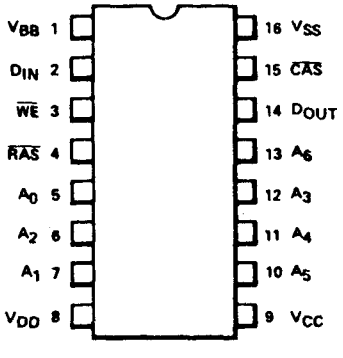


V_{SS} = Pin 16
V_{CC} = Pin 9
V_{DD} = Pin 8
V_{BB} = Pin 1

BLOCK DIAGRAM



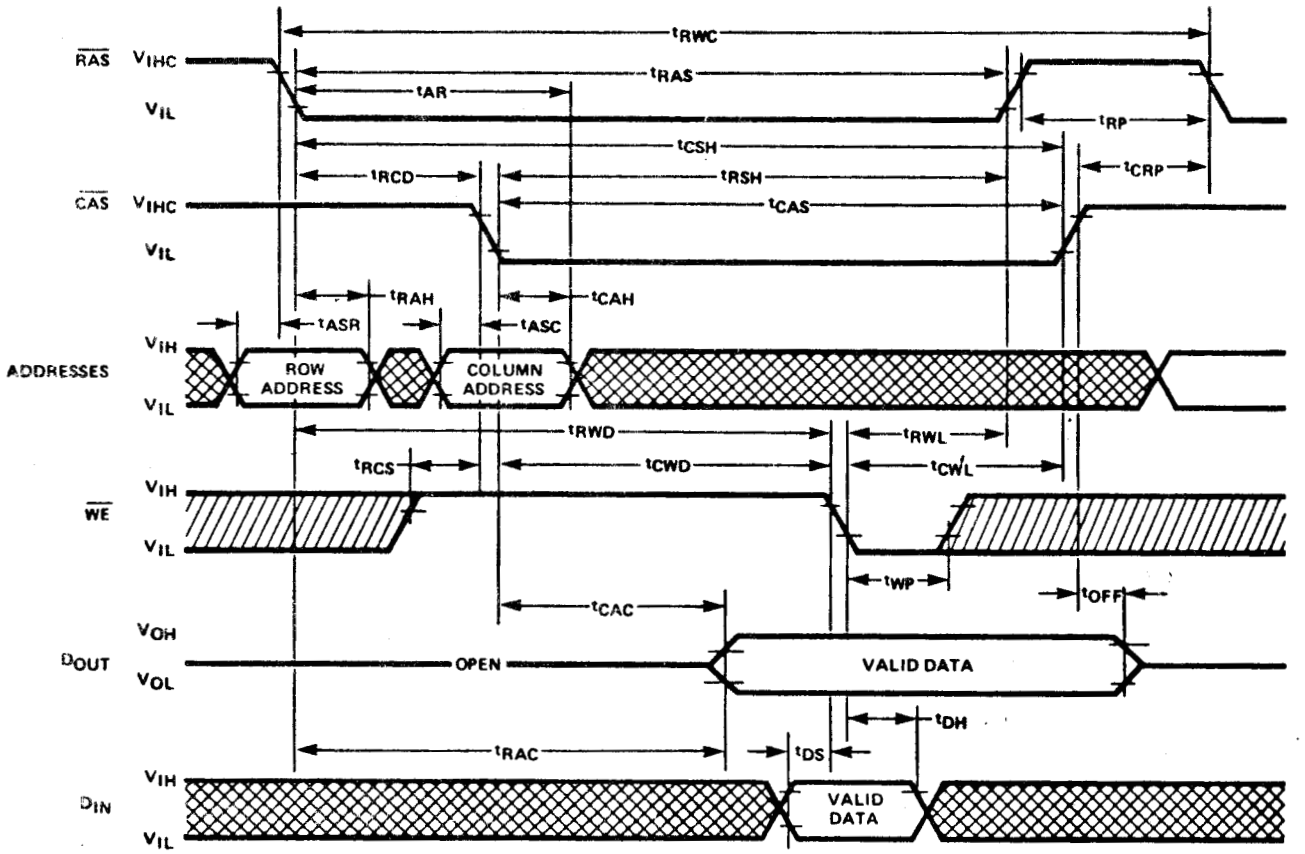
CONNECTION DIAGRAM
DIP (TOP VIEW)



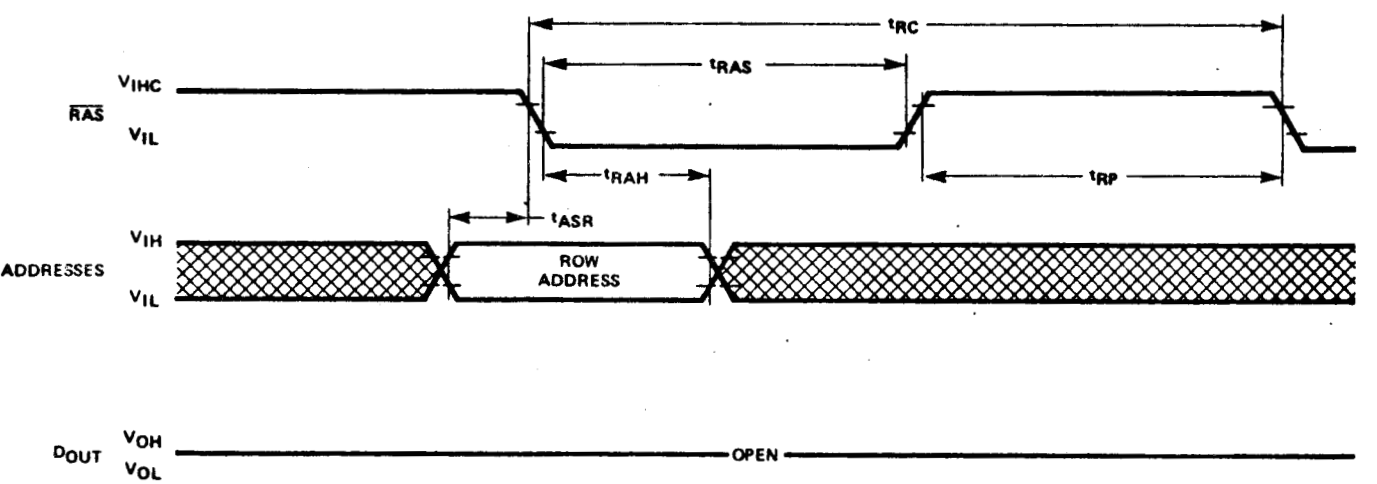
464 [illegible] MOUNTAIN VIEW, CALIFORNIA 94042 • (415) 962-5011 • TWX 910-379-6435

 Printed in U.S.A. 202-11-0001-047 15M

READ-WRITE/READ-MODIFY-WRITE CYCLE



"RAS ONLY" REFRESH CYCLE



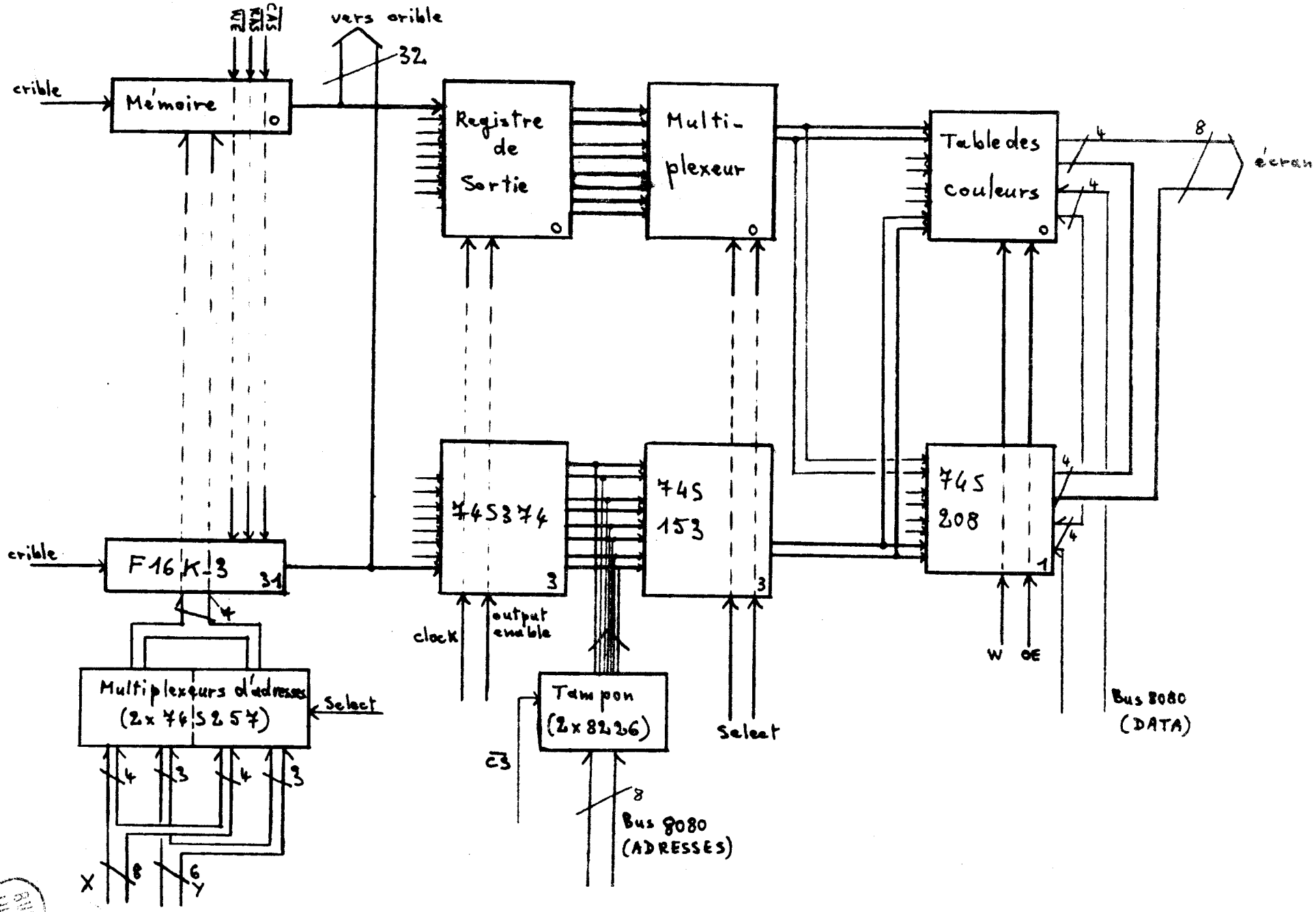
= May change in either direction

= May change HIGH-to-LOW

= May change LOW-to-HIGH

RECOMMENDED AC OPERATING CONDITIONS: Over Full Range of Voltage and Temperature and Voltage Range

SYMBOL	PARAMETER	PART NUMBER								UNITS	NOTES
		F16K-2		F16K-3		F16K-4		F16K-5			
		MIN	MAX	MIN	MAX	MIN	MAX	MIN	MAX		
t_{RC}	Random Read or Write Cycle Time	375		375		410		500		ns	7
t_{RWC}	Read/Write Cycle Time	375		375		515		625		ns	7
t_{PC}	Page Mode Cycle Time	170		225		275		325		ns	
t_{RAC}	Access Time from $\overline{RAS}$		150		200		250		300	ns	8,10
t_{CAC}	Access Time from $\overline{CAS}$		100		135		165		195	ns	9,10
t_{OFF}	Output Buffer Turn-off Delay	0	40	0	50	0	60	0	70	ns	11
t_{RP}	$\overline{RAS}$ Precharge Time	100		120		150		190		ns	
t_{RAS}	$\overline{RAS}$ Pulse Width	150	10k	200	10k	250	10k	300	10k	ns	
t_{RSH}	$\overline{RAS}$ Hold Time	100		135		165		195		ns	
t_{CSH}	$\overline{CAS}$ Hold Time	150		200		250		300		ns	
t_{CAS}	$\overline{CAS}$ Pulse Width	100	10k	135	10k	165	10k	195	10k	ns	
t_{RCD}	$\overline{RAS}$ to $\overline{CAS}$ Delay Time	20	50	25	65	35	85	80	105	ns	12
t_{ASR}	Row Address Set-up Time	0		0		0		0		ns	
t_{RAH}	Row Address Hold Time	20		25		35		80		ns	
t_{ASC}	Column Address Set-up Time	−10		−10		−10		0		ns	
t_{CAH}	Column Address Hold Time	45		55		75		80		ns	
t_{AR}	Column Address Hold Time Referenced to $\overline{RAS}$	95		120		160		185		ns	
t_T	Transition Time (rise and fall)	3	35	3	50	3	50	3	50	ns	13
t_{RCS}	Read Command Set-up Time (RMW)	0		0		0		0		ns	
t_{RCH}	Read Command Hold Time	0		0		0		0		ns	
t_{WCH}	Write Command Hold Time	45		55		75		130		ns	
t_{WCR}	Write Command Hold Time Referenced to $\overline{RAS}$	95		120		160		240		ns	
t_{WCS}	Write Command Set-up Time	−20		−20		−20		0		ns	14
t_{WP}	Write Command Pulse Width	45		55		75		130		ns	
t_{RWL}	Write Command to $\overline{RAS}$ Lead Time	60		80		100		130		ns	15
t_{CWL}	Write Command to $\overline{CAS}$ Lead Time	60		80		100		130		ns	15
t_{DS}	Data In Set-up Time	0		0		0		0		ns	16
t_{DH}	Data In Hold Time	45		55		75		130		ns	16
t_{DHR}	Data In Hold Time Referenced to $\overline{RAS}$	95		120		160		260		ns	
t_{CRP}	$\overline{CAS}$ to $\overline{RAS}$ Precharge Time	−20		−20		−20		0		ns	
t_{CP}	$\overline{CAS}$ Precharge Time (Page-mode)	60		80		100		120		ns	
t_{RF}	Refresh Period		2.0		2.0		2.0		2.0	ms	
t_{CWD}	$\overline{CAS}$ to $\overline{WE}$ Delay	70		95		125		180		ns	17
t_{RWD}	$\overline{RAS}$ to $\overline{WE}$ Delay	120		160		200		290		ns	17



MEMOIRE DE RAFRAICHISSEMENT

2114
1024 X 4 BIT STATIC RAM

	2114-2	2114-3	2114	2114L2	2114L3	2114L
Max. Access Time (ns)	200	300	450	200	300	450
Max. Power Dissipation (mw)	525	525	525	370	370	370

- High Density 18 Pin Package
- Identical Cycle and Access Times
- Single +5V Supply
- No Clock or Timing Strobe Required
- Completely Static Memory
- Directly TTL Compatible: All Inputs and Outputs
- Common Data Input and Output Using Three-State Outputs
- Pin-Out Compatible with 3605 and 3625 Bipolar PROMs

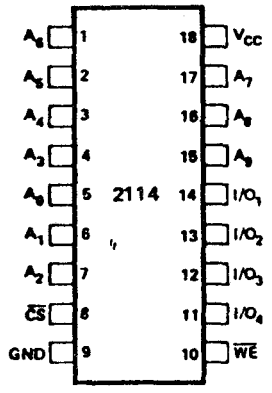
The Intel® 2114 is a 4096-bit static Random Access Memory organized as 1024 words by 4-bits using N-channel Silicon-Gate MOS technology. It uses fully DC stable (static) circuitry throughout — in both the array and the decoding — and therefore requires no clocks or refreshing to operate. Data access is particularly simple since address setup times are not required. The data is read out nondestructively and has the same polarity as the input data. Common input/output pins are provided.

The 2114 is designed for memory applications where high performance, low cost, large bit storage, and simple interfacing are important design objectives. The 2114 is placed in an 18-pin package for the highest possible density.

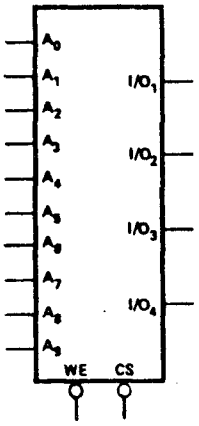
It is directly TTL compatible in all respects: inputs, outputs, and a single +5V supply. A separate Chip Select ($\overline{CS}$) lead allows easy selection of an individual package when outputs are or-tied.

The 2114 is fabricated with Intel's N-channel Silicon-Gate technology — a technology providing excellent protection against contamination permitting the use of low cost plastic packaging.

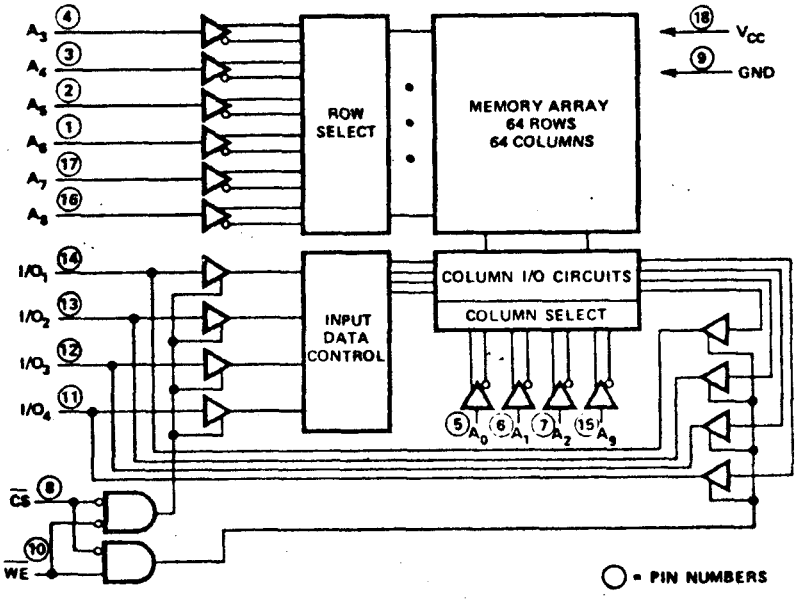
PIN CONFIGURATION



LOGIC SYMBOL



BLOCK DIAGRAM



PIN NAMES

A_0–A_9	ADDRESS INPUTS	V_{CC} POWER (+5V)
$\overline{WE}$	WRITE ENABLE	GND GROUND
$\overline{CS}$	CHIP SELECT	
I/O_1–I/O_4	DATA INPUT/OUTPUT	

MARCH 1977

A.C. CHARACTERISTICS $T_A = 0°C$ to $70°C$, $V_{CC} = 5V \pm 5\%$, unless otherwise noted.

READ CYCLE [1]

SYMBOL	PARAMETER	2114-2, 2114L2 Min.	2114-2, 2114L2 Max.	2114-3, 2114L3 Min.	2114-3, 2114L3 Max.	2114, 2114L Min.	2114, 2114L Max.	UNIT
t_{RC}	Read Cycle Time	200		300		450		ns
t_A	Access Time		200		300		450	ns
t_{CO}	Chip Selection to Output Valid		70		100		120	ns
t_{CX}	Chip Selection to Output Active	20		20		20		ns
t_{OTD}	Output 3-state from Deselection		60		80		100	ns
t_{OHA}	Output Hold from Address Change	50		50		50		ns

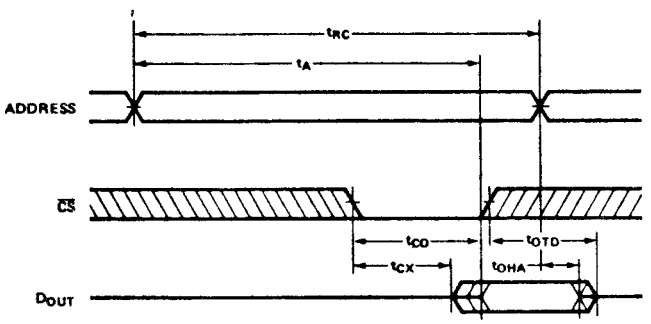
WRITE CYCLE [2]

SYMBOL	PARAMETER	2114-2, 2114L2 Min.	2114-2, 2114L2 Max.	2114-3, 2114L3 Min.	2114-3, 2114L3 Max.	2114, 2114L Min.	2114, 2114L Max.	UNIT
t_{WC}	Write Cycle Time	200		300		450		ns
t_W	Write Time	120		150		200		ns
t_{WR}	Write Release Time	0		0		0		ns
t_{OTW}	Output 3-state from Write		60		80		100	ns
t_{DW}	Data to Write Time Overlap	120		150		200		ns
t_{DH}	Data Hold From Write Time	0		0		0		ns

NOTES:
1. A Read occurs during the overlap of a low $\overline{CS}$ and a high $\overline{WE}$.
2. A Write occurs during the overlap of a low $\overline{CS}$ and a low $\overline{WE}$.

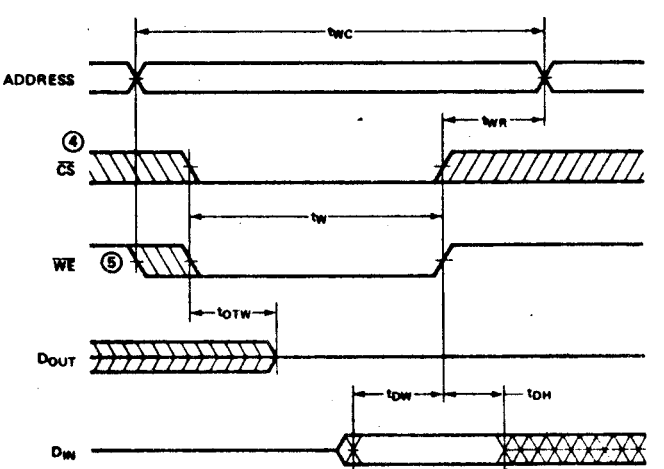
WAVEFORMS

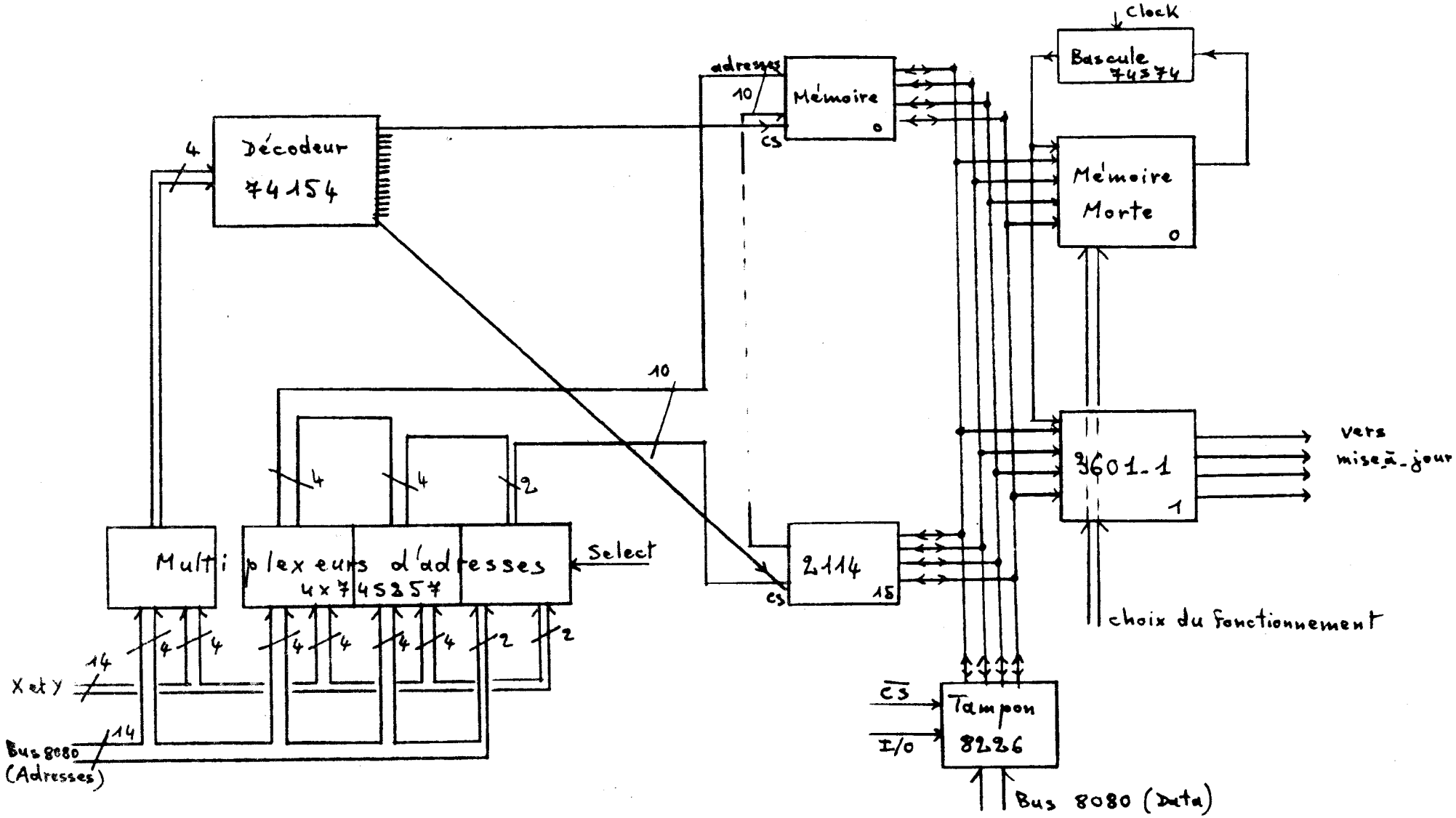
READ CYCLE ③



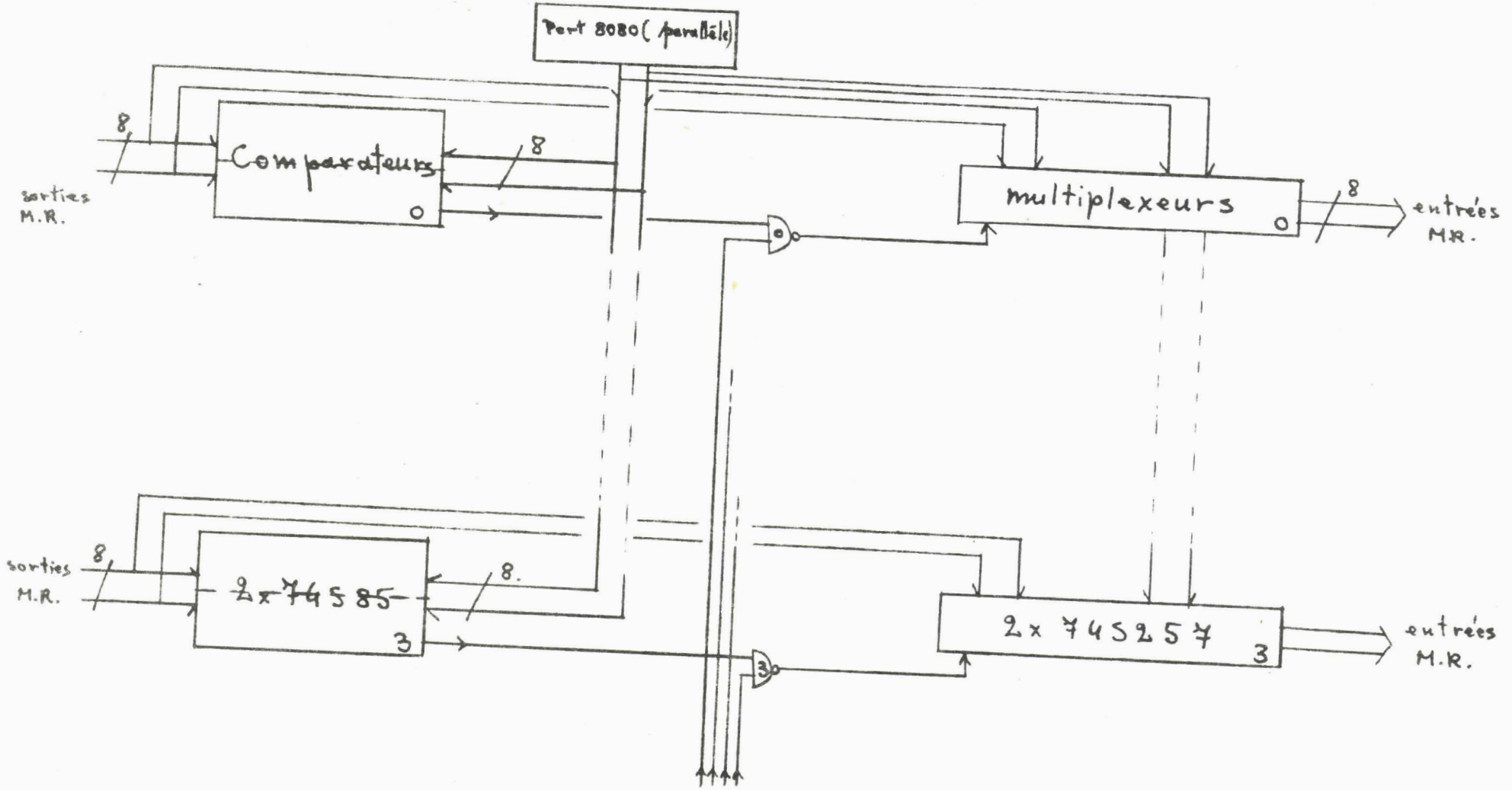
NOTES:
③ $\overline{WE}$ is high for a Read Cycle.
④ If the $\overline{CS}$ low transition occurs simultaneously with the $\overline{WE}$ low transition, the output buffers remain in a high impedance state.
⑤ $\overline{WE}$ must be high during all address transitions.

WRITE CYCLE





CRIBLE



PROCESSEUR DE MISE-A-JOUR