MASTER EN SCIENCES INFORMATIQUES

MACCOI : une extension au logiciel graphique interactif PICASSO

Piren, Sylvain

Award date:
1985

Awarding institution:
Universite de Namur

INSTITUT D'INFORMATIQUE
FNDP NAMUR

LABORATOIRE D'INFORMATIQUE
UNIVERSITE DE LILLE I

MACCOI:

UNE EXTENSION AU LOGICIEL GRAPHIQUE INTERACTIF PICASSO

MEMOIRE PRESENTE PAR
SYLVAIN PIREN
EN VUE DE L'OBTENTION
DU DIPLOME DE
LICENCIE ET MAITRE
EN INFORMATIQUE

ANNEE ACADEMIQUE
1984/85

TABLE DES MATIERES

TABLE DES FIGURES

FIGURE	PAGE

MACCOI: UNE EXTENSION AU LOGICIEL GRAPHIQUE PICASSO

Dans la suite de l'exposé, le sigle MACCOI (Manipulation, Codage, Compression et Optimisation d'Images) servira de référence globale à l'ensemble des fonctions réalisées dans le cadre de notre stage.

INTRODUCTION GENERALE

Depuis les temps les plus reculés, l'homme s'est servi d'images pour améliorer la communication avec ses congénères (illustration de parties de chasse, descriptions de stratégies de guerre, etc.). Les dessins étaient l'outil d'aide par excellence aux problèmes techniques (cf. Archimède, Léonard de Vinci).
Dès les débuts de l'informatique, des chercheurs se sont efforcés de créer des outils afin de pouvoir utiliser l'ordinateur comme moyen de communication graphique, de façon à produire, de nos jours, une multitude de logiciels graphiques.

Ces dernières années, on a vu sortir ces produits des laboratoires de recherches pour trouver une utilisation dans le domaine public et industriel. On peut s'en rendre compte assez facilement de par la prolifération des jeux vidéo dans le domaine public des loisirs et de sigles tels que E.A.O (Education Assistée par Ordinateur), dans le domaine public aussi, moins des loisirs peut-être , C.A.O. (Conception Assistée par Ordinateur) ou F.A.O (Fabrication Assistée par Ordinateur) dans un domaine plus technique, désignant des outils dans lesquels les composantes graphiques prennent une part importante.
En ce qui concerne la C.A.O., utilisée essentiellement dans l'industrie automobile, électronique ou d'aviation, elle s'est vue ouvrir un nouveau champ d'application voici quelques années.

En effet, des artistes (peintres, décorateurs, architectes etc.), séduits par les multiples possibilités que leur offrait l'informatique, ont exprimé le désir de se servir de l'ordinateur dans leur vie professionnelle.
C'est ainsi que des logiciels de C.A.O. artistique, appelés "palettes", tels que QUANTEL, AURORA ou PICASSO ont vu le jour.

La palette PICASSO (Peinture Interactive en Couleurs ASSistée par Ordinateur) étant non seulement un objet de recherche en informatique, mais aussi un produit destiné à être commercialisé, le but de notre travail était de lui adjoindre un certain nombre de fonctions (*)

le rendant plus compétitif au marché des palettes électroniques.

(*) On entend par fonction une particularité de la palette électronique, à ne pas confondre avec la notion de fonction en analyse fonctionnelle ou organique

Pour les besoins de la cause, notre travail s'articulera en trois parties:

- la première vise à mettre en évidence les objectifs des principaux champs d'informatique traitant des données graphiques (reconnaissance de formes et compréhension des images, traitement d'images, infographie interactive (*) et C.A.O.), d'introduire le système graphique PICASSO et de le situer par rapport à ces champs;

- la deuxième partie, plus importante, introduit des besoins non satisfaits par le système tel qu'il est exposé dans [CHARROUF 84], donc avant notre venue et expose, par fonction désirée, un survol de solutions proposées dans la littérature (si littérature il y a), les choix qui nous ont guidés vers une solution et la solution envisagée elle-même.

- la troisième et dernière partie de l'exposé fait état des différences essentielles entre une palette telle que PICASSO et des palettes plus évoluées telles que QUANTEL et propose une évolution possible de MACCOI.

Etant donné que le présent exposé s'adresse tout aussi bien à des gens désireux de trouver une introduction au sujet des palettes électroniques en général et du système PICASSO en particulier qu'à des informatitiens intéressés par la mise en oeuvre des solutions proposées, nous avons adjoints une partie ANNEXE reprenant l'analyse faite au départ de l'implantation des solutions ainsi que les algorithmes généraux établis.

(*) Notre travail s'adressant surtout à des lecteurs français, on utilisera par la suite le terme 'infographie interactive', traduction de l'expression anglaise 'interactive computer graphics

Je me réserve le reste de cette introduction pour exprimer ma gratitude aux personnes sans lesquelles ce mémoire ne serait pas.
Ainsi donc, j'exprime ma reconnaissance au professeur V. CORDONNIER qui m'a accordé une place de stagiaire dans le laboratoire d'informatique de l'Université de LILLE I, à messieurs M. MERIAUX, chargé de recherche au CNRS et Cl. CHERTON, professeur à l'institut d'informatique des FNDP, pour l'aide et l'asistance qu'ils m'ont prêtées durant la réalisation de mon travail. Je n'oublie pas Kh. CHARROUF, L. ALLAIN, B. SZELAG et tous ceux du laboratoire d'informatique pour leurs conseils précieux, me permettant ainsi de mener à bien mon travail.

PARTIE I: CADRE GENERAL DE MACCOI

INTRODUCTION

Tout travail appartient probablement à l'une des deux catégories suivantes: il est soit innovateur, soit prolongeur.
Nous parlerons d'un travail innovateur s'il est à la base d'un domaine nouveau, encore mal connu, voire inconnu, tel que le système SKETCHPAD crée en 1963 par I.E. Sutherland [SUTHERLAND 63] ou, dans une certaine mesure, les travaux concernant les systèmes experts, aujourd'hui.
D'un autre côté, nous désignerons par le terme "prolongeur" un travail se construisant à partir de bases solides, tels que les travaux en infographie interactive réalisés de nos jours.

Il nous semble évident que cette classification est arbitraire, peu de travaux, peut-être aucun, n'appartenant entièrement à l'une ou l'autre de ces deux classes. Pourtant, d'après notre définition, il est difficile de réaliser un travail innovateur en l'espace de quelques mois. Le nôtre ne faisant pas exception à la règle, il appartient à la catégorie des travaux prolongeurs.

C'est pour cette raison que la partie présente a pour but de mettre en évidence les bases sur lesquelles repose le logiciel MACCOI.

Chapitre 1 : CADRE DE PICASSO

Introduction

En nous plaçant dans la partie de l'informatique dans laquelle on manipule des images (représentation graphique d'un objet ou de plusieurs objets formés d'un ensemble de taches colorées) ou des dessins (id. mais constitué d'un ensemble de traits), nous nous apercevons très vite qu'un tel cadre serait beaucoup trop large pour pouvoir être pris comme référence.

On scindera donc cette partie de l'informatique en plusieurs champs, selon la classification utilisée dans [BRADY 82], et on attribuera à chaque champ ainsi défini un type d'image (*) le caractérisant (sans pour autant être exclusif).

La notion de "type d'image" est extraite de [LUCAS 82] et de [MERIAUX 79].

(*) Pour la commodité de l'emploi, on ne fera plus la distinction entre "dessin" et "image".

1.1. Types d'images existants

1.1.1. L'image photographique

Ce sont "toutes les images provenant d'un système optique ou électronique" [MERIAUX 79] tels que appareils photographiques ou scanners.

1.1.2. L'image programmée ou interactive

Les images programmées sont celles dont la constitution est fixée par programme et dont l'affichage ne sert qu'à visualiser, sans modification possible, l'image dont la représentation interne est donnée.

Les images interactives, par contre, sont établies en mode conversationnel entre un homme et un ordinateur, l'utilisateur pouvant donc manipuler l'image interactivement.
Ce deuxième type d'images peut encore être subdivisé:

1.1.2.1. L'Image graphique

On entend par images graphique une image visant à mettre en évidence les relations existantes entre diverses informations, non nécessairement graphiques.

1.1.2.2. L'Image animée

Ce sont des images dynamiques, comme leur nom l'indique déjà.

1.1.2.3. L'Image figurative

Ce sont celles communément utilisées en C.A.O. pour donner une vision abstraite de l'objet concret à réaliser.

1.1.2.4. L'Image non figurative

On désigne par ce terme les images "artistiques", non techniques. Ce sont surtout des images interactives de ce type qu'un utilisateur produira à l'aide de PICASSO.

Le fait que les frontières entre certains de ces champs ne soient pas très nettes ont conduit à la classification suivante:

1.2. Champs de traitement graphique de l'information

1.2.1. Reconnaissance de Formes/Compréhension d'Images

Les images typiquement manipulées en Reconnaissance de Formes et en Compréhension d'Images sont des images de type photographique t.q. des radiographies médicales, des photos de satellites, images issues du système visuel d'un robot etc..
On peut définir l'objectif de la Reconnaissance de Formes comme suit:
"The basic problem in pattern recognition is the classification of the various patterns, on the basis of a suitable set of measurements. The choice of such a set is of critical importance for the solution of the problem" [SAMI 73].
Dans le cadre de notre travail, une différentiation poussée entre Reconnaissance de Formes et Compréhension d'Images ne présente pas de grand intérêt. Signalons seulement que la différence principale réside dans le fait qu'en Reconnaissance de Formes, on travaille de manière générale directement sur l'image qui, en plus, appartient souvent à une classe restreinte de scènes possibles (il existe des analogies entre les images à analyser), alors que l'intérêt principal en Compréhension d'Images est de construire des descripteurs d'images et de manipuler, de traiter ces représentations symboliques.
Une telle description contient souvent deux parties majeures [FIRSCHEIN 72]:

- le contexte de la description, qui décrit l'impact émotionnel de l'image sur l'observateur et
- la description elle-même, contenant d'une part des renseignements plus objectifs (altitude de prise de la photo, angle de vue) et d'autre part, des faits subjectifs, décrits en langage naturel, concernant l'arrangement de la photographie en général ainsi que des structures sémantiques indiquant les relations entre les différents objets de la photographie.

On voit de suite que toute la partie de description subjective ne présente pas un vif intérêt dans le cas de la reconnaissance d'une scène par un robot p.ex.

1.2.2. le Traitement d'Images

S'il est possible de regrouper sous ce vocable tous les traitements possibles et imaginables concernant les informations graphiques, nous nous limiterons, dans la suite de notre exposé, aux objectifs suivants:

- la transmission des images (de T.V., de satellite)
- le stockage de ces images
- la restauration de celles-ci et
- leur amélioration.

Le dernier point concerne principalement le filtrage du "bruit" présent dans l'image. Celui-ci peut cependant revêtir plusieurs formes:
ce peut être de l'information non présente dans l'image originale (p.ex. dû à la transmission de l'image à travers les couches atmosphériques) ou bien de l'information inutile dans certains cas de traitement comme la "mise en évidence de zones polluées sur photographies aériennes.

Retenons encore que des expériences ont montré que l'ajout de bruit, créé artificiellement, par génération pseudo-aléatoire, peut faciliter la transmission de l'image sous forme compacte. Le bruit ainsi créé est filtré à la réception puisqu'il n'ajoute aucune information utile à l'image transmise. Cf. [ROBERTS 62] à ce sujet.

Sur la base des exemples cités, on pourrait croire que les seules images traitées en traitement d'images soient celles du type photographique; or il n'en est rien. Nous verrons dans la suite que les problèmes sous-jacents au traitement d'images apparaissent dans bon nombre d'applications travaillant essentiellement sur des images synthétiques, c-à-d produites sur ordinateur à l'aide d'outils spécialisés.

Les problèmes traités dans les deux classes (3 champs) qui viennent d'être présentées ont été fortement liés lors des premiers travaux qui y étaient consacrés. Ces classes ont donc des bases communes (problèmes liés à la détection de bords, de régions dans une image).

Le champ qui sera présenté dans la suite se trouve à part, les objectifs étant sensiblement différents. Ceci n'empêche pas des recouvrements entre eux.

1.2.3. Infographie Interactive/Conception Assistée par Ordinateur (C.A.O.)

Avant d'aborder une étude quelque peu détaillée de PICASSO, nous tenons à présenter encore les champs les plus importants sur lesquels repose le système PICASSO, c-à-d ceux de la C.A.O. et de l'Infographie Interactive.

En effet, PICASSO étant présenté comme un produit de C.A.O. artistique, il s'agit en premier lieu de définir le terme de C.A.O. en général.
Nous donnerons une définition de D.T. Ross, trouvée dans [GIAMBIASI ..]:

"La C.A.O. est une technique dans laquelle l'homme et l'ordinateur sont rassemblés pour la solution de problèmes techniques en une équipe qui allie étroitement les meilleures qualités de chacun d'eux, de telle manière que l'équipe travaille mieux que chacun séparément".

En bref, il s'agit donc de trouver la solution à une certaine catégorie de problèmes (conception automobile, de circuits intégrés, d'avions ...) en utilisant la C.A.O. comme moyen technique, comme outil. Cet outil complexe est essentiellement composé de trois fonctions:

1. une fonction de communication
   qui réalise l'interface entre l'homme et la machine. C'est elle qui constitue la partie visible du système de C.A.O., permettant à l'utilisateur de s'abstraire des problèmes informatiques.
2. la fonction de structuration des données
   Fonction d'archivage et de gestion des données du système; on y distingue les données statiques, uniquement utilisées par le concepteur (bibliothèque d'objets), et les données dynamiques, décrivant l'objet en cours de conception.
3. la fonction de traitement
   Cerveau du système de C.A.O., cette fonction assure la gestion de tout le processus de conception, en s'appuyant sur une base d'algorithmes.

La C.A.O. a, parallèlement à l'informatique, engendré plusieurs générations de systèmes, on en est actuellement à la troisième, la quatrième étant en évolution dans les laboratoires de recherches.
Toutes ces générations se distinguent en général par une assistance meilleure, une indépendance accrue entre fonctions et mêmes applications, seule la fonction de traitement étant liée à l'application.

Essayons dès à présent de mettre en évidence les relations existantes entre C.A.O. et Infographie Interactive.

L'outil que représente la C.A.O. est un outil très complexe, il est donc tout naturellement basé sur des outils plus simples, faisant office de 'primitives'. Ces outils appartiennent au champ de l'Infographie Interactive, du moins pour ce qui est de la fonction de traitement et éventuellement de communication, et au champ du traitement d'images dans le cas de la fonction de gestion des données.

Selon [BRADY 82], l'objectif principal de l'Infographie Interactive est d'afficher des informations graphiques: "considerable attention has been given to representing points, edges, surfaces and

volumes to facilite display. The geometry of perspectives and parallel (or orthographic) projections has been studied in detail".

Cet objectif correspond assez bien aux problèmes traités dans [FOLEY 82] et [NEWMAN 81]. La fonction de traitement est ainsi traitée en profondeur, la fonction de communication seulement de manière superficielle et les intérêts à la fonction de structuration des données ne portent que sur les données dynamiques. Pour donner une vision quelque peu globale sur les problèmes traités, nous allons essayer de dégager une structure générale pour les systèmes d'infographie interactive.

Comme notre travail ne concerne que l'aspect logiciel des systèmes graphiques interactifs, nous n'exposerons des aspects matériels que ceux propres à PICASSO, dans le chapitre suivant. Le lecteur intéressé par les divers matériels mis en oeuvre en Infographie Interactive en trouvera une description exhaustive dans [FOLEY 82], [NEWMAN 81] et [MORVAN 76].

1.2.4. Structure des logiciels graphiques interactifs (L.G.I.) [LUCAS 82] (cf. fig. 1.2)

On peut schématiser l'utilisation d'un programme d'application en mode conversationnel, avant de presenter la structure générale d'un L.G.I.. (cf. fig. 1.1)

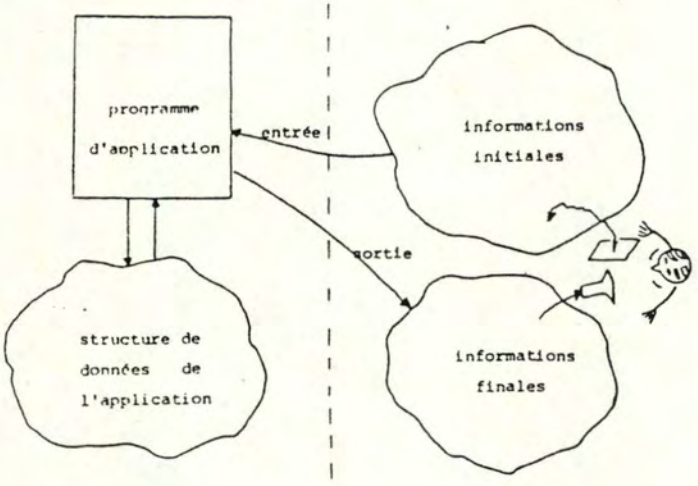


FIGURE 1.1: SCHEMA GENERAL D'UN PROGRAMME
D'APPLICATION EN MODE CONVERSATIONNEL
[LUCAS 82]

Un opérateur introduit, à l'aide de moyens d'entrée divers (souris, tablette graphique, photostyle) des informations dans l'ordinateur. Celles-ci seront pris en compte par un programme d'application qui, à l'aide de structures de données particulières, fournit un résultat sur un écran vidéo, une table traçante ou tout autre moyen de sortie.

Comme, dans la plupart des cas, les images finales fournies par le programme d'application, car ce sont elles qui nous intéressent à cet instant, sont d'un niveau assez bas (points,segments de droite), on intercale des logiciels graphiques entre le moyen d'entrée/sortie et le programme de visualisation.

On peut classer ces logiciels dans 3 catègories distinctes:

- le logiciel de description qui extrait des informations de la structure de donnees de l'application et les codifie dans le but d'obtenir une representation graphique du resultat de ce codage, appelè scène.

  Ce codage peut se faire à deux niveaux: au niveau du logiciel de description des donnees (dynamiques) ou au niveau de la fonction de structuration des donnèes statiques. C'est ce que nous allons voir dans la partie suivante où differentes formes de codages d'images seront etudiees. Le choix du procede de codage est souvent fonction de l'application, certaines donnees se codant mieux d'une maniere que d'une autre; il peut aussi dependre d'autres criteres comme de l'utilisation efficiente de la memoire, de la simplicite d'emploi ou encore de la complexite du procede de codage/decodage.

- le logiciel de preparation à la visualisation utilise la scène codee pour composer une scène bidimensionnelle sur un ecran de visualisation fictif. Le resultat en est un fichier graphique qui contient une description de l'image à produire.

- le logiciel elementaire, en fin de compte, a pour rôle de visualiser l'image à partir du fichier graphique elaboré à l'etape precedente.
  On dit que le logiciel elementaire produit une liste de visualisation et qu'il assure la gestion de l'ecran.

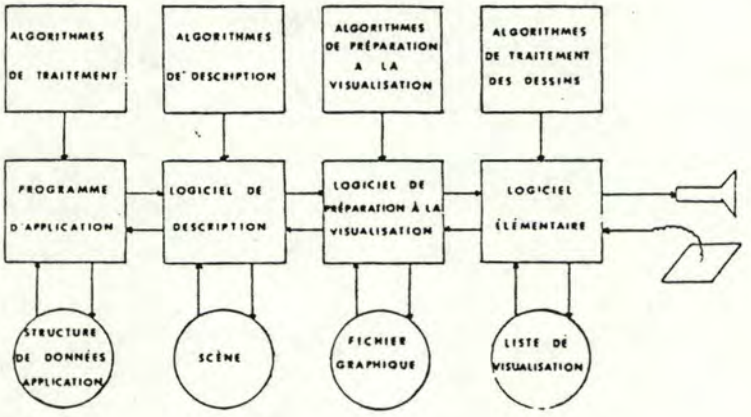


FIGURE 1.2: STRUCURE GENERALE D'UN LOGICIEL GRAPHIQUE INTERACTIF [LUCAS 82]

Dans la realite, les differents niveaux ne sont pas toujours separes de façon aussi nette. En effet, des applications purement graphiques n'ont pas besoin d'un niveau de description ou de stockage, le seul objectif etant la visualisation d'images.

Dans le cas où les logiciels sont integrès dans un systeme mono-application, le programme d'application peut être confondu avec les autres logiciels, si ce n'est que pour eviter une certaine redondance quant à la description des objets à manipuler.

Conclusion

Dès à présent, nous disposons d'assez d'éléments pour pouvoir présenter PICASSO et pour le placer par rapport aux champs que nous venons de voir.

Chapitre 2 : LE SYSTEME GRAPHIQUE INTERACTIF PICASSO

Introduction

Avant de présenter PICASSO, essayons tout d'abord de mettre en évidence les objectifs poursuivis. Si le système présente un côté recherche intéressant, il faut pourtant ne pas perdre de vue que PICASSO est aussi un produit commercial.

Bien que les objectifs initiaux étaient très généraux, visant la création d'un système facile à utiliser, offrant un large éventail de couleurs et de fonctions, utilisables par un grand nombre d'artistes d'horizons très divers, une étude de PICASSO [CEPE 84] a cristallisé ses principaux utilisateurs potentiels ainsi que son application principale. Ce seront des professionnels de la publicité (agences), des illustrateurs, des peintres, graphistes, décorateurs, personnes travaillant dans le textile et autres, utilisant PICASSO surtout (les agences de publicité dans ce cas précis) pour produire une ébauche, une maquette, présentable au client.
Ayant ceci en vue, il sera plus facile de s'accomoder aux diverses possibilités (ou impossibilités) de PICASSO.

Quelques préliminaires sont peut-être nécessaires pour bien comprendre son fonctionnement.

2.1. Preliminaires

2.1.1. La couleur

Nous allons passer ici sous silence des details de la theorie des couleurs. Une breve, mais bonne synthese peut etre trouvee dans [CHARROUP 84] et [FOLEY 82].

Ce qui nous interesse plus a ce stade, ce sont les modeles utilises pour representer les couleurs, comme ils vont refleter le mode de selection d'une couleur par une utilisateur.

2.1.1.1. Modeles objectifs

Le modele Rouge/Vert/Bleu (R.V.B.)

Le modele est appele R.V.B. parce que chaque couleur particuliere peut etre obtenue en melangeant dans les proportions correctes, les trois couleurs Rouge, Vert et Bleu, appelees couleurs primaires. Ces proportions sont mesurables et peuvent etre representees a l'aide de grandeurs physiques, C'est la raison pour laquelle un tel modele est dit objectif.

Le modele R.V.B. est notamment utilise pour l'affichage des couleurs dans les tubes a rayons cathodiques (C.R.T.).

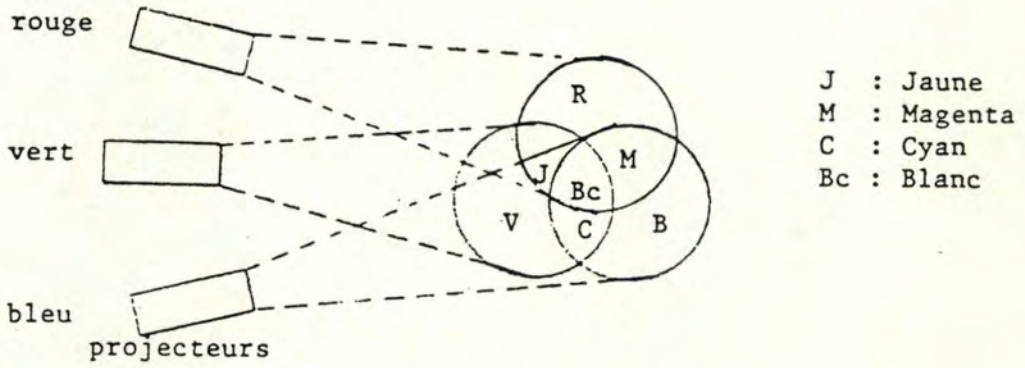


FIGURE 2.1 : SYNTHESE ADDITIVE RVB [CHARROUF 84]

Si de tels modeles peuvent etre tres utiles dans le cas d'un traitement technique de la couleur, dans les appareils de television, il n'en est pas de meme lorsqu'il s'agit d'un traitement humain, C'est dans cette optique que les modeles suivants ont ete developpes.

2.1.1.2. Modèles subjectifs

Le modèle Teinte/Intensité/Saturation (T.I.S.)

Nous avons beaucoup plus tendance à caractériser une couleur par des termes tels que couleur "vive", "terne", "sombre", "claire", "grisâtre", "pastel" etc., dénominations subjectives dans tous les cas.

PICASSO étant un produit de C.A.O. artistique, donc destiné à des peintres, décorateurs et autres créateurs, il serait souhaitable de disposer d'un modèle "subjectif", puisque basé sur des critères subjectifs de définition d'une couleur tels que ceux qu'on vient de citer.
De tels modèles (cf. fig. 2.3) ont été proposés par Smith, Munsell et Ostwald [FOLEY 82], [CHARROUF 84]. Ils se basent sur les trois aspects d'une couleur suivants: la teinte, l'intensité et la saturation. On parle du modèle T.I.S. en français ou H.L.S. (Hue, Lightness,Saturation) en anglais.

La teinte indique la "couleur" de la couleur, le caractère d'une couleur qui nous permet de la distinguer d'une autre. Nous disons qu'une couleur est "rougeâtre, jaunâtre, bleuâtre, ...".

L'intensité d'une couleur nous permet de distinguer les couleurs "claires" des couleurs "sombres".

Enfin, la saturation indique le degré de dilution d'une couleur dans le gris. Nous dirons dès lors qu'une couleur est "vive" ou "terne", selon que la saturation est grande ou non.

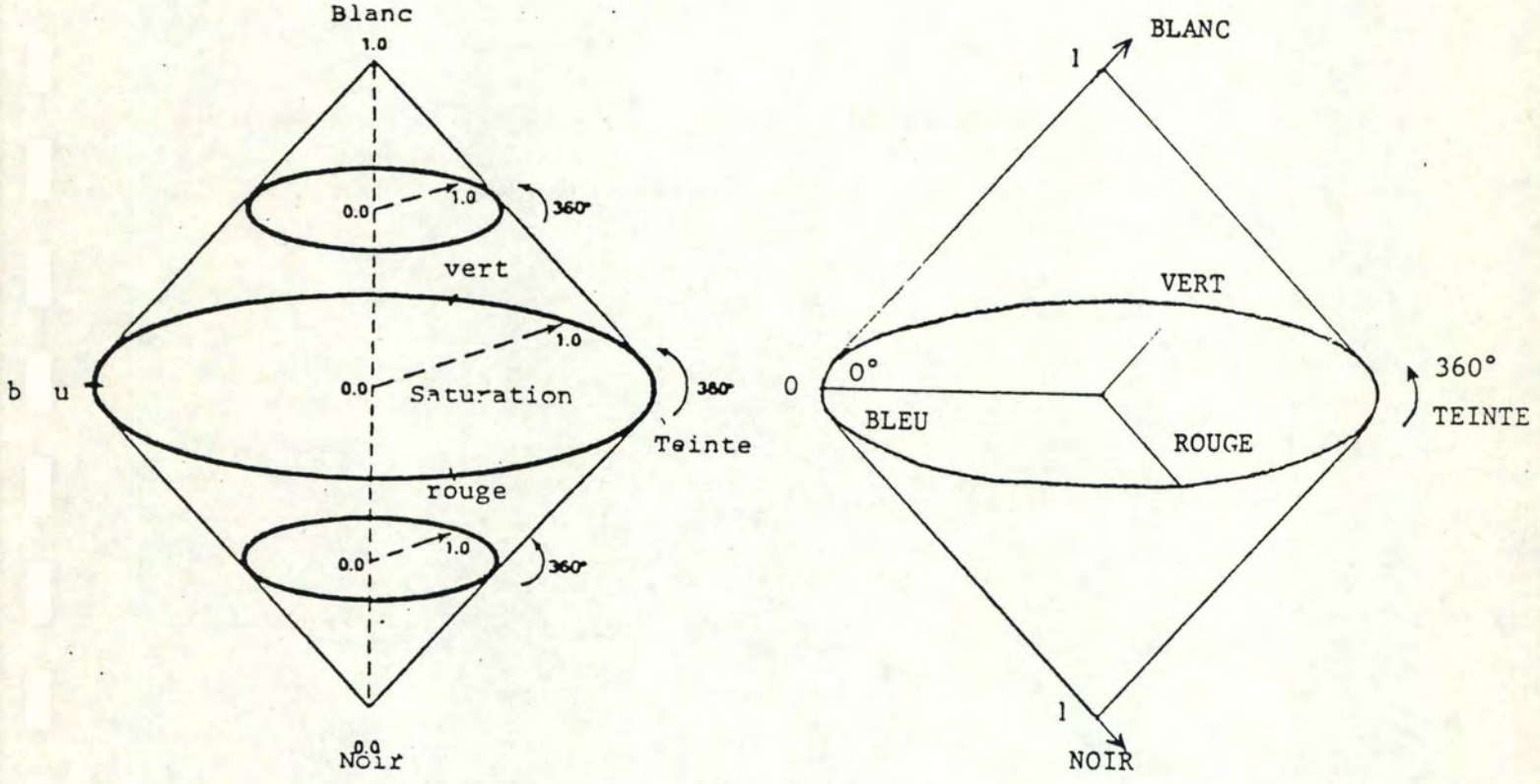
Le modèle T.I.S., associé à cette définition de la couleur, est représenté par un double cône, avec le blanc et le noir situés aux sommets respectivement supérieur et inférieur.
On peut se déplacer dans ce double cône selon trois directions différentes:
de l'extérieur vers l'intérieur, et vice versa, correspondant à une variation de la saturation,
d'un sommet vers l'autre correspondant à une variation de l'intensité et
autour de l'axe central, correspondant à une variation de la teinte.

Par convention, on dit que l'intensité et la saturation varient entre 0 et 1, alors que la teinte varie entre 0 et 360 degrés, le bleu se trouvant à 0 degrés.

Le blanc correspond à une intensité maximale, donc 1, et le noir à une intensité minimale, donc 0. Une saturation de 0 indique un gris total (on ne perçoit plus de teinte) et une saturation maximale n'est atteinte que si la couleur a une teinte pure, c-à-d dépourvue de blanc et de noir. La saturation et l'intensité sont donc dépendantes l'une de l'autre, alors que la teinte leur est indépendante.
L'état de saturation maximale est seulement atteint autour du cercle extérieur de la base des deux cônes.

Le modele Teinte/Blanc/Noir (T.B.N.)

Notons encore l'existence d'un modele analogue, le modele T.B.N., obtenu par changement d'axes à partir du modele precedent. Il est plus facile à utiliser, le fait d' ajouter du blanc ou du noir etant plus facilement assimilable qu'une saturation/desaturation - augmentation/diminution de l'intensite. Ces modeles sont representes ci-dessous:



T.I.S T.B.N

FIGURE 2.2 : LES MODELES T.I.S. ET T.B.N.
[CHARROUF 84]

2.1.2. Les menus

"A great many interactive graphical programs use menus for command and operation selection. A menu displays .. the full range of options, .. prevents the user from making selections outside this range and .. is flexible. In application prograns with many commands, it is normal to use a multilevel menu" [NEWMAN 79].

Une représentation naturelle pour les menus organisés hiérarchiquement est la structure arborescente, donc constituée de noeuds et de feuilles.
Nous dirons qu'un noeud sera représentatif d'une famille de commandes et qu'une feuille désigne toujours une commande simple. Une feuille ne peut évidemment être sélectionnée que si tous ses ascendants ont été choisis.
On appellera ordre toute sélection dans le menu, pouvant donc être aussi bien une famille qu'une commande.

Les noeuds peuvent représenter des familles inclusives ou exclusives, deux familles inclusives pouvant être actives en parallèle, alors que seulement soit l'une, soit l'autre de deux familles exclusives ne peut être active au même moment du traitement. Toutes les commandes d'une même famille sont évidemment exclusives, une seule commande pouvant être exécutée à un certain instant t de la session de travail.

2.1.3. Définitions

2.1.3.1. Pixel

Abbréviation de "Picture Element", un pixel (ou pel selon certains auteurs) représente l'information minimale pouvant être affichée sur un écran de visualisation. A chaque pixel est associée une valeur spécifiant la couleur ou le niveau de gris du point de l'image.

2.1.3.2. Mémoire de rafraîchissement

Nous distinguerons deux types différents de mémoires de rafraîchissement ou mémoires d'entretien selon que l'affichage se fait par balayage cavalier ou récurrent.

Dans le cas du balayage cavalier, utilisé essentiellement pour afficher des dessins au trait, le faisceau électronique est dirigé successivement vers les points d'articulation du dessin, sans cheminement préfixé.

Le balayage récurrent est celui utilisé dans les écrans de télévision. Le rayon cathodique parcourt l'écran complet à une

cadence d'environ trente fois par seconde.

La structure de la mémoire d'entretien ressort du mode de balayage utilisé.
Dans le premier cas, elle doit contenir une liste de points (coordonnées) constituant le dessin, tandis qu'elle doit pouvoir contenir l'information caractérisant un écran complet, même les parties non affectées par l'image, dans le deuxième cas, celui qui nous intéresse ici.
La mémoire d'entretien est dès lors organisée en une matrice de points représentant un écran fictif.

Si les dessins sont souvent en noir et blanc pur, la valeur d'un pixel pouvant alors être représentée par un seul bit, il en est tout autrement pour les images.
Elles sont formées de taches ayant en général soit différents niveaux de gris, soit plusieurs couleurs.
Un pixel est donc identifié par une coordonnée (X,Y) et une valeur de plusieurs bits; 256 couleurs différentes pouvant être représentées par un octet.

Dans le cas où plus de finesse dans les couleurs est nécessaire, il faudrait recourir à une représentation sur plus de bits; cependant, la taille de la mémoire augmenterait rapidement et il serait de plus en plus difficile d'afficher la mémoire à la cadence minimale de 25 fois par seconde.
Une solution à ce problème est donnée par l'utilisation d'une table de fausses couleurs.

2.1.3.3. Table de fausses couleurs (T.F.C.)

Nous venons de voir que le temps de lecture de la mémoire d'entretien impose une limite supérieure à sa taille et donc au nombre de couleurs différentes pouvant être représentées.

La solution proposée consiste à ne plus mettre la valeur d'une couleur directement dans le pixel, mais d'y mettre un indice vers une table, appelée Table de Fausses Couleurs qui, elle, peut contenir toutes les couleurs à afficher. Il est à noter en plus qu'une addition d'une nouvelle couleur dans la T.F.C. n'affecte pas la mémoire d'entretien. Le nombre de couleurs différentes possibles peut être très supérieur au nombre d'entrées de la T.F.C., sinon son utilisation n'aurait guère d'intérêt.

Ainsi, si chaque couleur primaire est codée sur n bits dans la T.F.C. et que celle-ci possède m entrées, m couleurs peuvent être affichées en même temps parmi les 2**(3n) possibles.

Le schèma suivant illustre le principe de fonctionnement d'une T.F.C.:

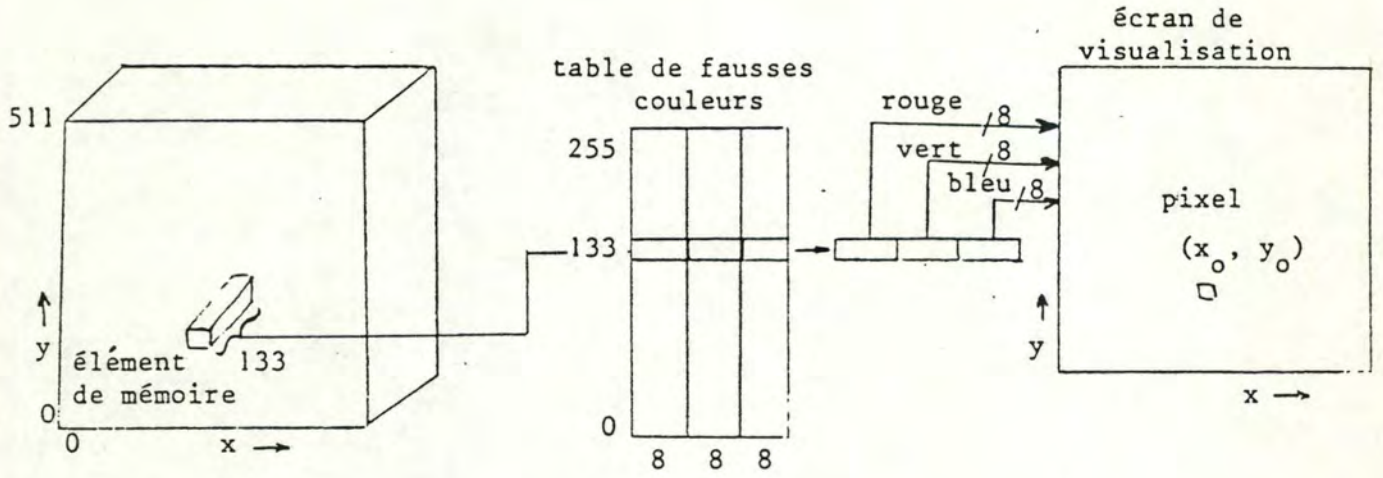


FIGURE 2.3 : PRINCIPE DE FONCTIONNEMENT D'UNE T.F.C. [CHARROUF 84]

2.1.3.4. Processeur graphique

Il nous a paru nècessaire de rappeler brièvement ce qu'est un processeur graphique et ce que sont ses fonctions principales. On peut affirmer qu'un processeur graphique est en gènèral un calculateur spècialisè, travaillant avec des instructions comme un processeur ordinaire, avec la difference que les codes operations dèsignent des actions graphiques. Celles-ci peuvent être des transformations geomètriques telles que translation ou rotation, des calculs directs sur les pixels (modification de la couleur) ou encore des manipulations diverses sur des images telles que des operations logiques entre images. En ce qui concerne le rafraichissement de l'ecran a l'aide de mèmoire d'entretien, c'est un processeur d'affichage qui s'en occupe, sur le plan fonctionnel du moins; en realite cependant, il s'agit tres souvent du même processeur graphique.

2.2. Le materiel utilise dans PICASSO

Expliquons la configuration utilisée à l'aide du schéma suivant:

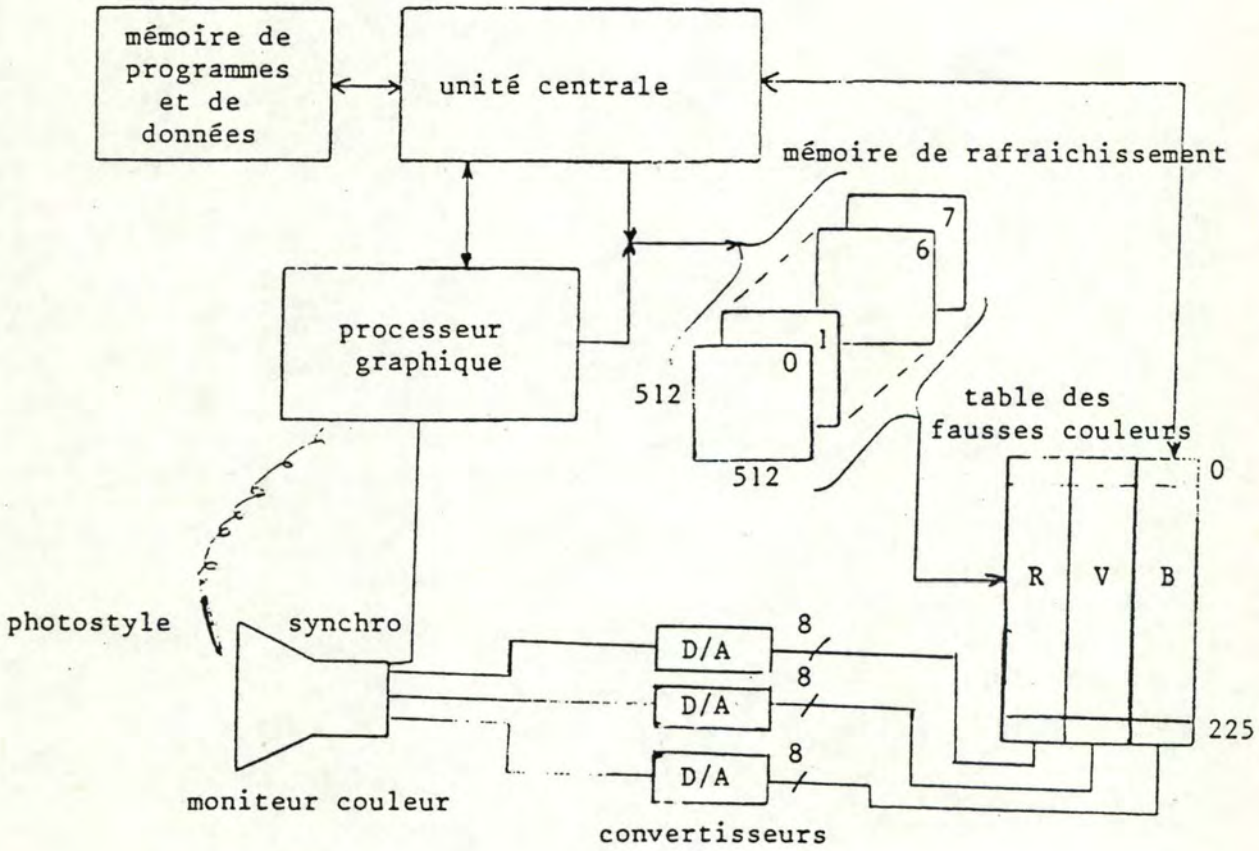


FIGURE 2.4 : CONFIGURATION MATERIELLE DE PICASSO [CHARROUF 84]

L'unité centrale est constituée d'un processeur 8088 (16 bits) et éventuellement d'un processeur 8087 faisant office d'unité arithmétique rapide pour les calculs exécutés en virgule flottante. Elle prend en charge l'acquisition et l'exécution des ordres donnés par l'utilisateur ainsi que le calcul des valeurs des pixels individuels.

Le processeur graphique joue le rôle d'interface entre l'unité centrale et l'écran, en écrivant dans la mémoire d'entretien ou dans la

T.F.C. des valeurs calculées par l'unité centrale et en lisant la mémoire d'entretien. Une autre fonction du processeur graphique est la gestion du photostyle, duquel il fournit les coordonnées du point désigné à l'unité centrale.

La mémoire de rafraîchissement est un tableau de 512 * 512 éléments, de 8 bits chacun. L'utilisateur dispose donc à chaque instant de 256 couleurs différentes. Comme la mémoire d'entretien ne fait pas partie de l'espace adressable de l'unité centrale, elle est accédée par port d'entrée/sortie. Du fait que, en plus, un pixel est caractérisé par 5 octets(2 + 2 pour les coordonnées, 1 pour l'indice vers la T.F.C.), l'opération d'écriture est coûteuse en temps. Il s'en suit qu'un remplissage complet de l'écran pixel/pixel nécessite approximativement 15 secondes.

La T.F.C. associée à la mémoire d'entretien contient 256 valeurs différentes codées sur 3 octets, affectées respectivement aux couleurs Rouge, Vert et Bleu. PICASSO offre ainsi (2**3)**8 = 2**24 couleurs différentes (presque 17 millions), dont 1536 teintes, à l'utilisateur.

L'écran de visualisation est à résolution moyenne: 512 * 512 pixels. Quatre signaux lui sont fournis, un signal de synchronisation et trois signaux provenant de la conversion digitale/analogique des éléments de la T.F.C.

Le seul moyen d'entrée actuellement utilisé est le photostyle (lightpen), qui permet de désigner directement un endroit sur l'écran; une "souris" est en cours d'implémentation.

Le point de vue matériel étant exposé, nous pouvons passer dès à présent à une présentation du logiciel mis en place pour réaliser PICASSO.

2.3. Le logiciel de PICASSO

Lors de la creation du système PICASSO, on a été soucieux de garder une analogie entre le système et les outils de travail d'un peintre, c-à-d une toile, un pinceau, des moyens de composition d'une couleur et des fonctions diverses, que nous allons exposer par la suite.
La surface de visualisation se présente à l'utilisateur comme un plan divisé de manière inégale en quatre parties, la première étant ce que nous appellerons par la suite "image" c-à-d la surface de dessin effective.
Les trois autres parties représentent les menus, inclusifs, sous-divisés en menu de la palette, des fonctions et du pinceau.

Le schéma suivant reprend cette subdivision de l'écran:

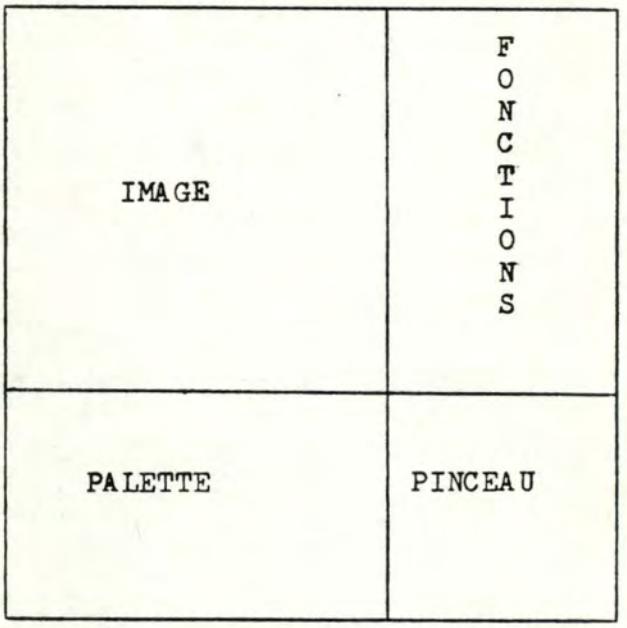


FIGURE 2.5 : SUBDIVISION DE L'ECRAN DANS PICASSO

Examinons d'abord le dernier menu:

2.3.1. Le pinceau

Il ne contient aucune famille, une modification de son épaisseur (PLUS/MOINS) étant une commande élémentaire par définition.
Tournons-nous dès à présent vers les deux menus restants, plus complexes que le pinceau: la palette et le menu des fonctions. Ce sont deux familles inclusives d'un noeud implicitement choisi (la racine de l'arbre):

2.3.2. La palette

La palette presentee dans le systeme PICASSO est basee à la fois sur le modèle T.I.S. et T.B.N.
L'utilisateur peut donc faire des modifications sur la teinte, l'intensité, la saturation, la quantité de blanc et de noir d'une couleur.
Le menu de la palette est constitué de deux parties:

1. 7 pavés de couleurs contenant initialement six couleurs différentes, espacées de 60 degrés au niveau des teintes (la couleur bleue est deux fois présente: à 0 et à 360 degrés):

BLEU	MAGENTA	ROUGE	JAUNE	VERT	CYAN	BLEU

Une action sur cette partie de la palette permet donc de modifier la teinte de la couleur courante.

2. Des commandes permettant diverses actions sur le cercle des teintes et sur les deux autres paramètres des modèles. Ce sont:
   - Modifications sur l'intensité, la saturation, le blanc et le noir:

PLUS / CLAIR
PLUS / VIF
MOINS \ BLANC
MOINS \ NOIR

Ces commandes sont globales pour TOUTES les teintes accessibles à un certain moment, influençant ainsi un cercle complet de teintes.
   - Modification du voisinage d'une teinte

VOISINES - RETOUR

Ces commandes permettent d'affiner ou de grossir le voisinage dune teinte particulière en affichant des teintes voisines immédiates (VOISINES) ou en supprimant ces teintes intermédiaires (RETOUR). Au niveau le plus bas, les nuances entre teintes ne seront plus perceptibles alors qu'au niveau le plus haut se trouvent les teintes initiales, éventuellement modifiées en intensité ou saturation.

   - Défilement séquentiel du cercle des teintes

PLUS \
MOINS / DEFILER

On peut définir le niveau de voisinage d'une teinte particulière comme étant constitué de l'ensemble des teintes du cercle espacées entre elles de n degrés. Il y a donc

autant de niveaux qu'il existe de distances différentes possibles. La commande DEFILER permet de visualiser toutes les teintes d'un même niveau en les faisant défiler soit vers la gauche, soit vers la droite. Cette commande n'a aucun effet au niveau le plus haut où toutes les teintes du niveau sont affichées.

Les différentes manipulations possibles sur la palette étant maintenant élucidées, nous passons à la présentation du menu des fonctions.

2.3.3. Le menu des fonctions (*)

Ce menu est organisé hiérarchiquement et ne contient que des familles et des commandes exclusives.
Au niveau supérieur se trouve une fonction (dessin en trait continu c-à-d à main levée) et quatre familles de fonctions.

La première désigne une famille d'actions sur les couleurs, telles que:

- MELANGE de la couleur courante avec une couleur sur l'image;
- REMPLISSAGE d'une tâche avec la couleur courante;
- mettre le FOND de l'image à la couleur courante;
- prendre comme couleur courante la COULEUR-SUR-L'IMAGE ou encore
- EFFACER, fonction qui permet d'effacer une partie de l'image, comme avec une gomme, le pinceau recevant la couleur du fond.

Une autre famille permet de dessiner des formes géométriques telles que

- SEGMENT,
- LIGNE BRISEE,
- TRIANGLE,
- CARRE,
- RECTANGLE,
- PARALLELOGRAMME,
- LOSANGE,
- CERCLE et
- ELLIPSE.

Il suffit de donner quelques points comme paramètres (en général 2) pour permettre à PICASSO d'executer la fonction et de dessiner la forme choisie.

La troisième famille permet de positionner une grille sur l'écran, dans le cas où il faut dessiner des formes à des endroits précis, assembler des blocs, etc.. Le pas de la grille (distance entre les points de celle-ci) est modifiable.

La dernière famille est particulière. En effet, les fonctions qui y sont reprises concernent essentiellement des ordres de sortie, telles que

- cadrer l'image pour prendre une PHOTO
- remettre la machine dans l'état initial (RESET)
- mettre du TEXTE sur l'image en commutant le moyen d'entrée (photostyle) vers un clavier alphanumérique.

(*) Dans la suite, on désignera par fonction une commande de ce menu

Conclusion

Nous avons essayé de définir le contexte général dans lequel se situe notre travail, aussi bien en ce qui concerne le macrocosme représenté par des champs tels que le traitement d'images, l'infographie interactive ou la conception assistée par ordinateur, que le microcosme défini par le système PICASSO.

D'après l'exposé qui vient d'être fait, le lecteur n'aura aucune difficulté d'admettre que PICASSO est un pur produit de C.A.O. et ainsi d'infographie interactive, outil de base de la C.A.O.

La partie suivante servira à donner une vue en détail de notre travail. Considéré comme extension au système tel qu'il vient d'être présenté, cette réalisation nous obligera de resituer PICASSO par rapport aux champs rappelés ci-dessus.

PARTIE II : LE LOGICIEL MACCOI

INTRODUCTION

Le système PICASSO, tel qu'il vient d'être exposé, présente essentiellement deux avantages et désavantages [CEFE 84]. C'est un produit très facile à utiliser, permettant une recherche rapide et aisée dans les couleurs mais dont la configuration matérielle et la fonctionnalité (nombre de commandes dans la terminologie que nous avons utilisée) est assez réduite par rapport aux autres produits présents sur le marché.

Comme nous venons de le voir, PICASSO n'offre pas la fonction de stockage et de gestion des données, essentielle dans un système de C.A.O. . Cette déficience est mise en évidence par [CEFE 84]. En effet, après avoir créé une image, l'artiste se voit contraint d'en faire éventuellement une photo et puis d'effacer l'image pour pouvoir en dessiner une autre.

Cette situation étant inacceptable, l'objectif principal du travail consistait donc en la réalisation d'un outil permettant le stockage d'une image, fonction si importante que tout le travail en découle.

Du fait que stockage et codage sont souvent intimement liés, le cahier des charges prévoyait une fonction de CODAGE/DECODAGE permettant de réduire l'espace mémoire occupé par une image. Qui dit stockage dit AFFICHAGE, sans lequel le stockage n'a aucun intérêt. Cette fonction nous a permis d'offrir un moyen de MISE-A-JOUR d'une image à l'utilisateur.

Toutes les fonctions citées tiennent à un fil. Pourtant, deux fonctions supplémentaires ont été reprises dans le cahier des charges, devenant seulement possible par l'existence même du moyen de mémorisation. Ce sont d'une part une fonction de correction automatique (ARRIERE) et d'autre part, une fonction d'optimisation d'une image. L'ensemble de ces fonctions vont être analysées dans les trois chapitres suivants.

Chapitre 3 : CODAGE ET COMPRESSION

Introduction

Souvent, la méthode de codage indique aussi comment il faut stocker, le codage n'étant qu'un moyen de stockage particulier. Selon ce principe, le chapitre suivant aura deux objectifs:

1. exposer plusieurs méthodes de codage trouvées dans la littérature
2. exposer et justifier la méthode de codage utilisée.

3.1. Procèdès de codages divers

Depuis presque les dèbuts de l'informatique, on a essayè de comprimer les donnèes à mèmoriser. Des solutions ètaient proposèes pour les images digitalisèes [WHOLEY 61] et la transmission d'images de tèlèvision [ROBERTS 62].

Ces techniques ètant très diverses et souvent adaptèes à des problèmes particuliers, on exposera une sèrie de mèthodes de codage dècrites dans la littèrature, en partant des types les moins adaptès à notre propos pour en arriver à ceux qui sont rèellement intèressants.

3.1.1. Grammaire de polygones [MERIAUX 79]

Ce type de codage, utilisè essentiellement en reconnaissance de formes et en analyse d'images, appartient à la famille des codes gèomètriques.
Pour pouvoir l'appliquer, il s'agit de dèfinir d'abord un alphabet, constituè par des symboles graphiques, p.ex. des formes gèomètriques simples telles que le triangle, le carrè ou le cercle. Ensuite doit-on disposer de règles de grammaire dèfinissant les diffèrentes façons d'assemblage des symboles èlèmentaires. Celles-ci peuvent être comparèes aux règles syntaxico-sèmantiques dèfinissant les constructions possibles d'une phrase à l'aide d'un l'alphabet des lettres dans une langue dèterminèe.

Dans ce type de codage, un parallèlogramme pourrait être reprèsentè de la manière suivante:

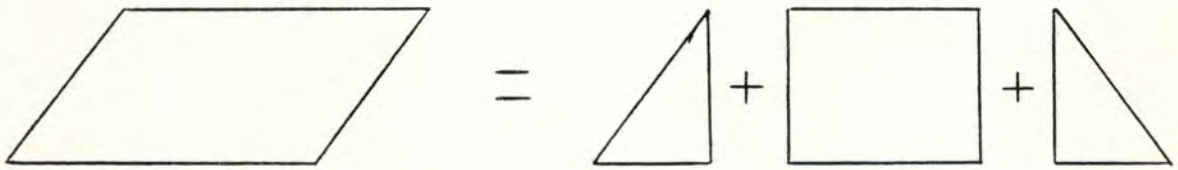


FIGURE 3.1 : EXEMPLE D'UTILISATION DU CODAGE PAR POLYGONES [MERIAUX 79]

On se rend compte qu'il y a de fortes limitations en ce qui concerne les objets constructibles, si on ne veut pas se heurter à des complications ènormes:
comment reprèsenter p.ex. le tracè à main levèe?
Un tel type de codage est donc tout à fait inadaptè à notre propos.

3.1.2. Changement d'espace [MERIAUX 79]

Dans une telle méthode de codage, on passe d'une image représentée sous forme matricielle dans l'espace bidimensionnel dans un espace dit des "fréquences" (par analogie avec la transformée de Fourier en électronique).

Des calculs analytiques tels que la transformée de Fourier bidimensionnelle ou spatiale ou encore la transformée de Walsh-Hadamard sont donc appliqués. Des détails sur de telles méthodes de codage peuvent être trouvés dans [ZAHN 72], [ROESE 77] et [ANDREWS 76]. De tels codages sont surtout utilisés en analyse d'images réelles, donc du type photographique, pour en filtrer le bruit. Ils sont inappropriés aux images de synthèse.

3.1.3. Approximations analytiques [MERIAUX 79]

Ce procédé est utilisé pour coder des images dont les éléments sont des courbes ou des surfaces.
On essaye alors d'approximer celles-ci par des calculs analytiques, en cherchant des coefficients numériques caractérisant de façon complète la courbe ou la surface. L'utilisation de fonctions polynomiales dont la précision dépend du nombre de coefficients utilisés en est la solution. Dans le cas où l'image ou un de ses constituants serait trop complexe, on la (le) subdivise pour approximer alors les éléments trouvés.

Cette méthode est largement utilisée en CAO lors de la représentation et de l'étude de formes de voitures ou d'avions.
Elle nécessite pourtant beaucoup de calculs et des améliorations existes (courbes de Bézier, approximations Spline).

Les images artistiques produites à l'aide de PICASSO rendent une utilisation de ce type de codage impossible.

3.1.4. Codage de Huffman [WELCH 84]

Dans ce cas, on s'efforce de traduire des données de longueur fixe en données de longueur variable en se basant sur la constatation que tous les symboles utilisés n'ont pas la même fréquence d'apparution. Les symboles les plus répandus sont codés avec le nombre le plus petit de bits. Une application directe en est la compression d'un texte anglais ou d'un programme source.

Lorsqu'on désire appliquer ce procédé, on part des types de redondances suivants:

1. distributions variables des symboles
   p.ex. la lettre "e" est la plus utilisée en anglais
2. répétitions de symboles
3. motifs à usage fréquent
   p.ex. le mot PERFORM dans un programme source COBOL
4. redondances positionnelles
   p.ex. l'apparition d'une ligne droite verticale sur un écran de visualisation.

En voulant utiliser un codage de Huffman, il faut
a) disposer de statistiques sur les distributions des symboles plus, éventuellement, de chaînes de caractères.
b) disposer d'une table de correspondance symbole/code.

Il est à remarquer qu'avec une table restreinte aux nombres d'occurences possibles d'un seul symbole (256 entrées pour 1 symbole de 8 bits), seul le premier type de redondance peut être éliminé.
Dans tous les autres cas, il faudrait une table de correspondance croissant selon la puissance de 2 (codage sur deux caractères => 2**16 = 64K entrées dans la table), ce qui est absolument prohibitif mais serait nécessaire dans le cas du codage d'une image de PICASSO. Il faudrait même disposer d'une table de 16M entrées, dans le cas d'un codage par couleur (16M couleurs différentes).

3.1.5. Run-length coding [CAPON 59]

[MERIAUX 79] en parle comme d'un cas particulier du codage par transitions, mais nous n'exposerons que la méthode du run-length coding, étant donné que c'est l'expression la plus utilisée dans la littérature.
En bref, cette méthode se base sur la constatation qu'une longue suite de symboles identiques peut être représentée de manière univoque par le couple suivant:

(identificateur du symbole, nombre d'occurences successives)

Le run-length coding a trouvé de larges applications dans le stockage d'images, celles-ci présentant une grande redondance à ce niveau. La méthode est pourtant inadaptée aux images photographiques, les variations de couleur étant trop nombreuses et aléatoires.

Très compacte lors de l'application aux images graphiques, elle est pourtant tout à fait inadaptée à une manipulation des images sous forme codée. En plus, le fait de travailler avec une table des fausses couleurs complique son utilisation, comme l'image à coder ne contient pas toutes les informations relatives aux couleurs utilisées.
Pourtant, cette méthode nous semble être celle utilisée dans la majorité des cas pour stocker des images, sur lesquelles on n'effectue alors pas de manipulations sous forme codée. Ceci a pour effet que bon nombre de méthodes de codage différentes sont basées sur le run-length coding; nous en exposerons quelques unes des plus intéressantes par la suite.

3.1.6. Codage cellulaire

C'est la seconde et dernière méthode de codage géométrique dont nous parlerons. Dans ce type de codage d'images, celles-ci sont divisées en cellules de petite taille qui seront codées individuellement par une méthode telle que le run-length coding.

La taille du code est fonction de la complexité de l'image c-à-d du nombre de tâches différentes, de la taille d'une cellule et de l'encodage de la cellule individuelle utilisé.

3.1.7. Codage predictif

Il s'agit d'une méthode étudiée pour la première fois en 1950 par Elias. On peut dire que, dans cette méthode, "the dependance inherent in the data can be removed by a good predictor ("fonction de prédiction" N.d.l'A.) that transforms the original data into a form such that successive data symbols are nearly independant of each other. The transformed data can then be encoded by techniques applicable to independant sources " [KOBAYASHI 74].

Il ressort de cette définition que le codage se fait au moins en deux phases, la première rendant les données le plus indépendantes possibles, la deuxième codant alors le résultat. Il est à noter que la première phase ne produit aucune compression.

Comment fonctionne cette méthode?
On part du principe que, dans une image, la valeur (couleur) d'un pixel est fonction de ses prédécesseurs et qu'il y a moyen de trouver une fonction de prédiction pouvant "prédire" la valeur d'un pixel, la meilleure fonction étant celle qui minimise le taux d'erreurs.
Le schéma suivant montre quels pixels influencent celui dont on veut prédire la valeur:

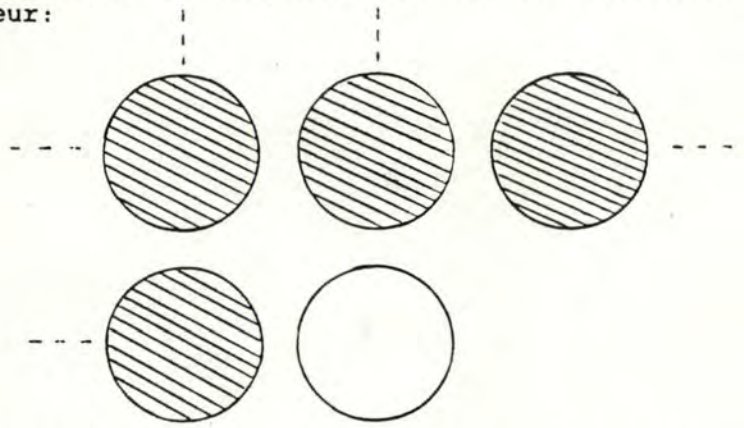


FIGURE 3.2: INFLUENCE DES PIXELS AVOISINANTS EN CODAGE PREDICTIF [KOBAYASHI 74]

Lors du parcours de l'image, on applique la fonction et l'on constitue en parallèle une matrice des erreurs, de dimension identique à celle contenant l'image originale, où chaque élément sert à déterminer s'il y a eu erreur ou non lors de l'application de la fonction de prédiction à l'élément. Si nous supposons que "0" indique l'absence et "1" la présence d'erreur, la valeur d'un pixel dans la matrice des erreurs pourra être représentée par un seul bit. En appliquant maintenant un codage tel que le run-length coding à la matrice des erreurs, on peut obtenir un bon taux de compression, si la fonction de prédiction est bonne.

Il existe des fonctions de prédiction fixes, utilisées lorsqu'on dispose de statistiques sur les données et des fonctions adaptatives, utilisées surtout si les données ne sont pas connues.
Le codeur et le décodeur seront normalement implémentés par matériel et des traitements sur les objets codés sont impossibles. Selon [WHOLEY 61], la compression obtenue est moyenne, de l'ordre de 1:3, de meilleurs taux pouvant être obtenus en appliquant des techniques

utilisées en reconnaissance de formes (approximations), qui ont pourtant l'inconvénient de ne pas garder tous les détails d'une image.

3.1.8. Codage de Freeman [MERIAUX 79]

La méthode de codage présentée ici s'applique le mieux au dessin puisqu'elle code les contours d'un objet.
Pour ce faire, Freeman s'est servi de la notion de "direction".

Supposant qu'on dispose d'une grille bidimensionnelle et qu'on parte d'un point quelconque déterminé par l'intersection de deux mailles, on peut alors affirmer qu'il y a au plus 8 voisins à un pixel et que, dès lors, 8 directions sont seulement possibles pour accéder à l'un quelconque de ces pixels à partir du pixel dont on part. Freeman a donné des codes à chacune des directions de la manière suivante:

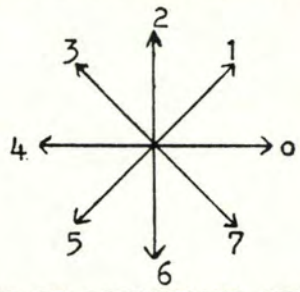


FIGURE 3.3 : CODES DE DIRECTIONS UTILISES DANS FREEMAN [MERIAUX 79]

La figure suivante :

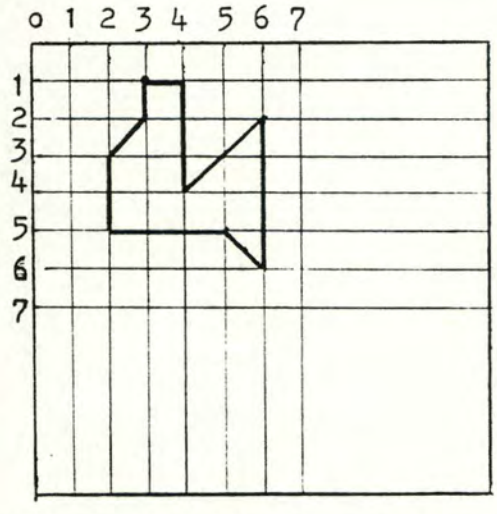


FIGURE 3.4 : EXEMPLE DU CODAGE DE FREEMAN

pourra être codée comme suit:

(3,1) 6,5,6,6,0,0,0,7,2,2,2,2,5,5,2,2,2,4

Sans facteur de répétition, nous avons un dessin à 18 côtés dont le codage prendrait (7 + 7) + (18 * 3) = 68 bits. Ce codage est adaptable aux images en spécifiant pour chaque contour la couleur à l'intérieur. "Le codage de Freeman est utilisable de même manière dans

un espace à trois, voire quatre dimensions" [MERIAUX 79].
Il est actuellement utilisé dans le codage des images graphiques (non photographiques) qu'il permet de structurer et de manipuler facilement (translation, rotation, zoom).

3.1.9. Transformation de Hadamard [MEYER 83]

On ne présentera pas la méthode elle-même mais une dérivée, proposée dans [MEYER 83], utilisée pour stocker des images médicales (images de scanner p.ex.) dans le cadre d'un PACS en médecine (Picture Archieving and Communication System).

L'image originale étant représentée sous forme matricielle, on en produit quatre sous-matrices de tailles identiques égales à un quart de la taille de la matrice de départ.
Dans la première sous-matrice, un élément contient la somme de quatre pixels (formant un carré) de la matrice de départ; les trois autres sous-matrices contiennent des sommes et des différences des mêmes éléments par pixel.

Le schéma suivant illustre le fonctionnement.

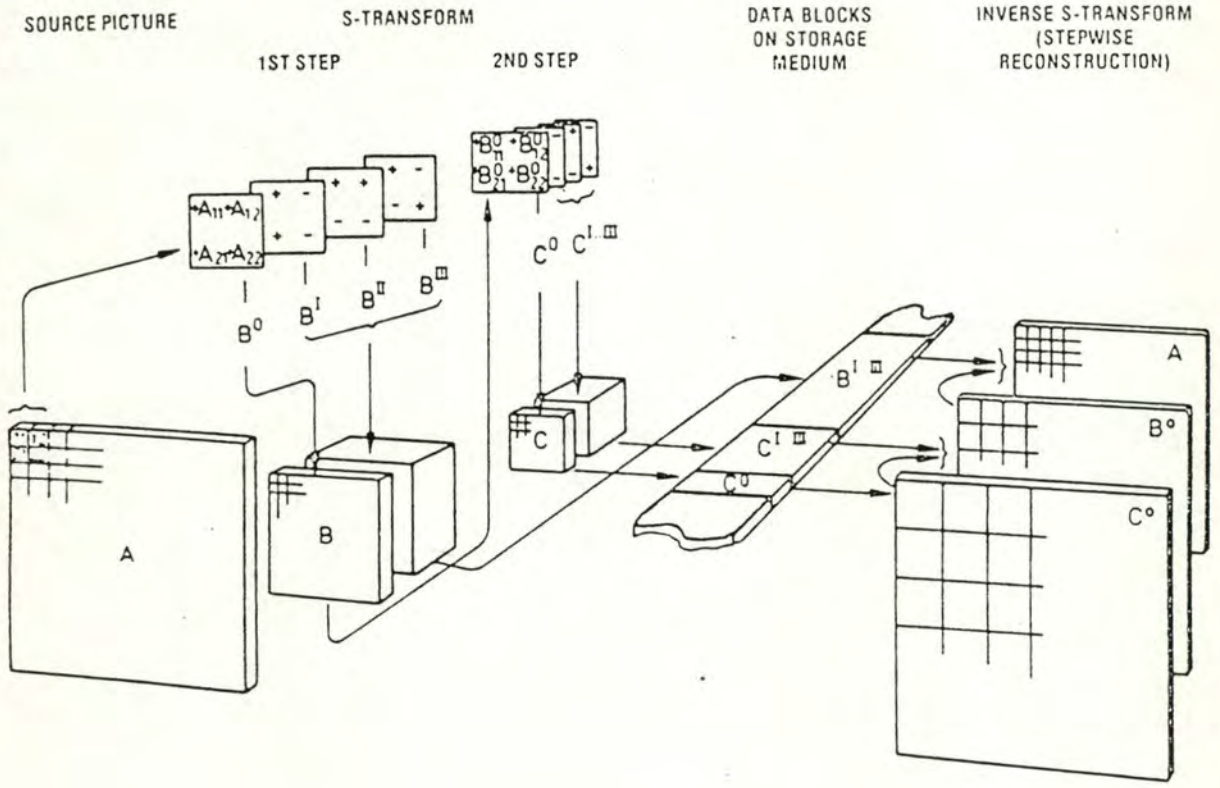


FIGURE 3.5 : TRANSFORMATION DE HADAMARD [MEYER 83]

La matrice B0 représente l'image de départ à résolution réduite de

moitié; B1, B2, B3 contiennent les détails manquants.

Ce procédé peut être itéré autant de fois que l'on le veut, en réduisant à chaque coup la résolution d'un facteur de 2. On stocke toutes les matrices des détails ainsi que la dernière matrice "image". Si on veut réduire la taille de stockage, ces matrices peuvent être encodées selon une méthode classique.

Cette approche est très intéressante si l'on dispose d'écrans à résolutions différentes. On réduit l'image de départ à la résolution minimale et, lors du réaffichage, on peut, selon la résolution de l'écran de visualisation, afficher de plus en plus de détails.

3.1.10. Quadtree [WOODWARK 84] [OLIVER 83] [DYER 80]

Le procédé possède les mêmes avantages que le codage précédent, mais est encore mieux du fait que l'image originale est disponible à tout instant, sans que l'on doive faire de longs calculs afin de la retrouver.

Dans cette approche, la structure de données est un arbre, comme le nom l'indique. La représentation de l'image est obtenue en la divisant récursivement en quatre parties égales jusqu'à ce qu'on obtient une zone de même couleur. Ce seront là les feuilles de l'arbre, les noeuds représentant tous des parties d'image à plusieurs couleurs. La racine de l'arbre est formée par l'image entière.

Si l'on sauve dans un noeud la valeur moyenne des couleurs des quatre noeuds (ou feuilles) de niveau inférieur lui étant attachés, il y a moyen, comme pour la codage par transformation de Hadamard, d'obtenir une représentation à résolution multiple. En effet, un noeud non-terminal représente un point de l'image à demi-résolution de celle des noeuds lui étant attachés.

Un exemple de quadtree très simple est présenté sur le schéma suivant:

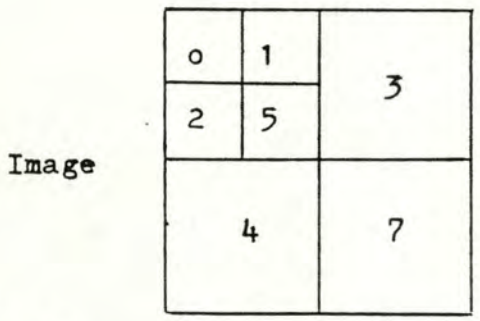


FIGURE 3.6 : EXEMPLE DE QUADTREE [OLIVER 83]

L'interprétation du code en est la suivante:
le traitement s'effectue dans l'ordre suivant: quadrant inférieur

gauche, supérieur gauche, inférieur droit et supérieur droit.

Les couleurs sont représentées sur 3 bits, un quatrième bit indique s'il s'agit d'une feuille (0) ou d'un noeud (1).
A toute valeur d'un noeud, il s'agit donc d'ajouter 16, le bit indiquant un noeud ou une feuille étant celui de poids le plus élevé.

Un autre avantage des quadtrees est la possibilité de manipuler des images sous forme codée. [OLIVER 83] proposent des algorithmes de manipulation d'images (rotation,translation, remplisage d'une tâche par une couleur, ...).

Remarque: Signalons qu'il y a des rapports théoriques certains entre les points 3.1.2, 3.1.9 et 3.1.10, que le cadre du présent exposé ne permet pas de mettre en évidence.

3.2. Stockage/Codage dans PICASSO

3.2.1. Traitement des ordres dans PICASSO

Le fonctionnement de PICASSO peut être représenté par l'ordinogramme suivant:

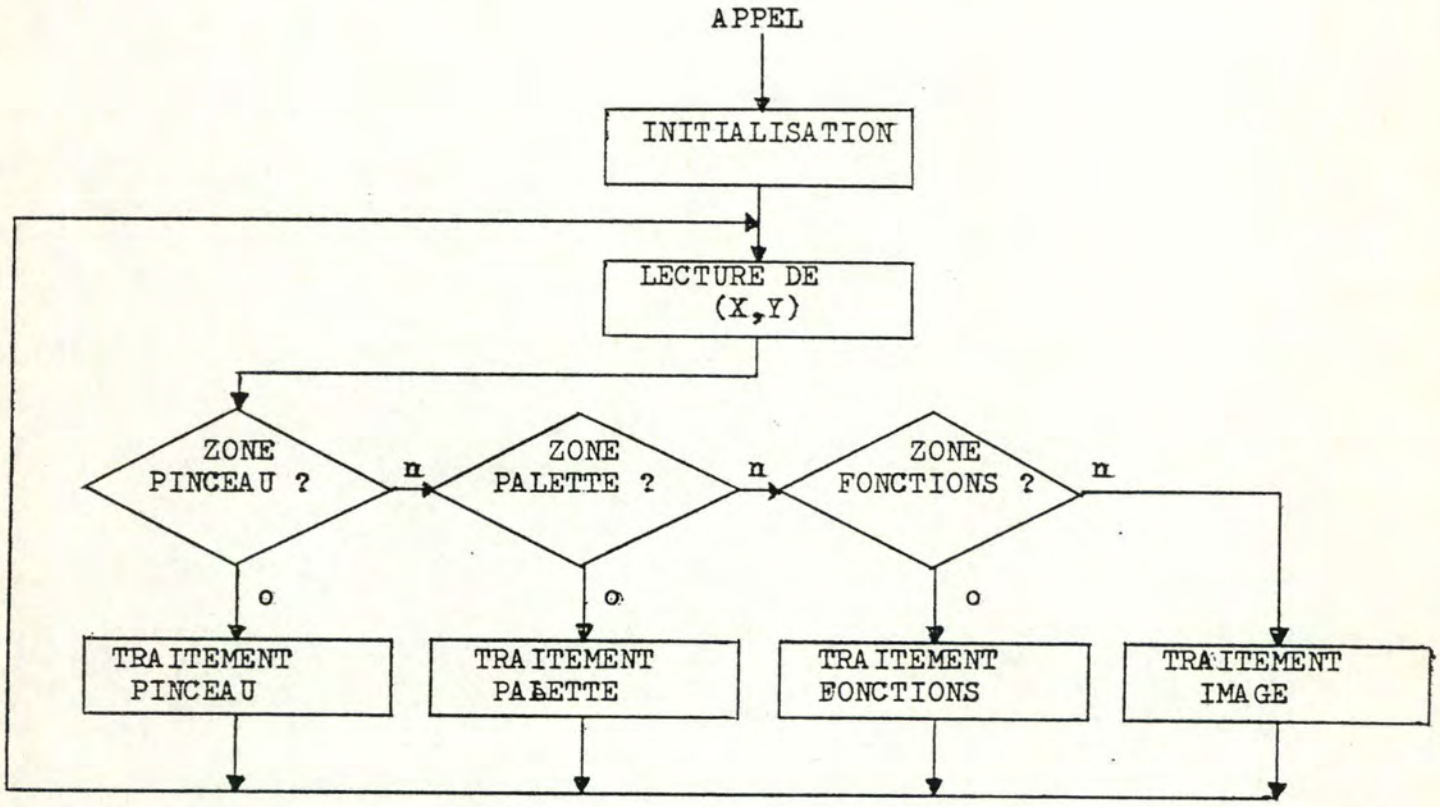


FIGURE 3.7 : ORDINOGRAMME GENERAL DE PICASSO

On a volontairement omis de mettre une instruction conditionnelle, PICASSO n'offrant pas de sortie logicielle, l'utilisateur devait éteindre la machine pour terminer sa session de travail.
L'ordinogramme nous montre que PICASSO traite des ordres données par l'utilisateur en pointant une zone particulière de l'écran avec le photostyle. Celui-ci transmet les coordonnées en (X,Y) associées au point lu. PICASSO décode alors les coordonnées comme appartenant à l'une des quatre zones principales composant l'écran: la palette, le menu des fonctions, le pinceau et l'image.

Pour ce faire, il tient en mémoire tout le contexte nécessaire à l'exécution d'un ordre comme p.ex. la fonction, la couleur, le pinceau à utiliser lors d'un pointage de la zone "image".

La plupart des fonctions nécessitent qu'un ou deux paramètres, fourni par l'utilisateur en pointant la zone image. Les fonctions de la famille COULEUR sont généralement à un paramètre et les fonctions de la famille FORMES à deux. PICASSO se charge donc des calculs nécessaires pour l'exécution de la fonction désirée.

3.2.2. Stockage d'images

Soit I une image et O_{t_i} un ordre donné à l'instant ti du temps de production de I.
D'après le paragraphe précédent, on peut dire que I est obtenu en exécutant les O_{t_i} de t0 à tn. I est donc définie complètement par la suite des ordres suivants:

$$I = O_{t0}, O_{t1}, \ldots, O_{tn}$$

Cette suite d'ordres définit non seulement l'image finale de manière univoque, mais aussi tout le processus de construction inhérent à l'image.

Cette propriété est très intéressante dans la mesure où on veut étudier le comportement d'un artiste vis-à-vis de PICASSO ou, de manière générale, sa façon de construire une image (quelles couleurs ou fonctions, quels pinceaux sont utilisés, comment l'artiste commence-t-il une image ...).
Une telle étude est actuellement en cours à Lille.

Une deuxième propriété de cette structure d'image est qu'elle permet de visualiser cette construction lors d'un réaffichage de l'image ce qui entraine en même temps un effet d'animation qu'il y a moyen d'exploiter.
Ces deux propriétés ne se retrouvent pas dans une image stockée sous sa forme finale, matricielle, beaucoup plus pauvre en informations. S'y ajoute encore que l'utilisation d'une T.F.C. rendrait un stockage impossible, dans la mesure où la matrice ne contient qu'une image finale "réduite", étant donné qu'elle ne contiendra que 256 couleurs parmi les 16 millions possibles. Pour stocker la "vraie" image finale, il faudrait représenter chaque point de l'image sur 24 bits, ce qui triplerait le volume de stockage, sans parler des complications se posant lors du réaffichage de l'image, en devant passer des 24 aux 8 bits de la mémoire d'entretien. La matrice aurait alors une taille de 512*512*3 = 768 Koctets, ce qui est énorme.

Finalement, on peut encore noter encore que la phase de stockage proprement dite, que nous distinguerons de la phase de codage, entraine déjà une compression à deux niveaux:

1. Comme PICASSO interprète lui-même les ordres reçus par l'utilisateur (pouvant être une personne lors de la création ou une machine lors d'un réaffichage), il n'est pas nécessaire de représenter des couleurs, ni sur 8, ni sur 24 bits.
2. Alors que le stockage sous forme matricielle implique une taille fixe, quelle que soit la complexité de l'image produite, la taille de l'image sous forme d'une suite d'ordres est évidemment variable selon la complexité.

Si nous envisageons une taille de 4 octets par point, nous pouvons aisément calculer le gain obtenu lors de la production d'une image très simple: soit un carré bleu sur fond rouge, en prenant comme couleurs celles présentes après l'initialisation du système.

Les actions à effectuer sont:

A) production du carré
   1) choisir la famille
   2) choisir la fonction
   3) donner deux paramètres

B) changement de la couleur de fond
   1) sélectionner la couleur rouge
   2) choisir la famille des couleurs
   3) choisir la fonction FOND
   4) donner un paramètre

Notons que B.1 pourrait se trouver aussi bien après B.2 ou B.3 du fait que les couleurs et les fonctions appartiennent à des familles inclusives.

Nous avons du donner 8 paramètres, donc 32 octets. Il est évident qu'un stockage sous forme matricielle, même avec un codage approprié, serait tout à fait prohibitif dans ce cas précis. Si de telles images de très petite taille sont fort rares, nous verrons dans la partie suivante qu'elles peuvent être très intéressantes dans certains cas et de ce fait, devenir plus fréquentes.

Mais il serait certes plus opportun de s'occuper d'images plus réalistes en ce qui concerne leur taille et complexité. Au moment de la rédaction, nous ne disposons malheureusement pas encore de statistiques sur des images créées par des artistes sur PICASSO. Néanmoins, quelques observations peuvent être faites :

Il est certain que des artistes issues de différentes catégories professionnelles produisent des images d'une complexite très variable, un publiciste créant des images differentees de celles d'un graphiste ou d'un architecte. Cette différence dépend essentiellement de deux facteurs qui sont la quantité de dessin à main levée, plus coûteux en place mémoire que l'utilisation de formes géométriques, et le nombre de recherches dans les couleurs, celles-ci augmentant elles aussi la complexité de l'image.

L'utilisation du photostyle fatiguant en plus les yeux, à cause de la proximité de l'écran, on peut raisonnablement supposer un nombre d'heures limité investi à la création d'une image, bien que la mise-à-jour de l'image pourrait naturellement accroître sa taille.

D'autre part, les images créées au laboratoire par des artistes donnent une idée sur l'occupation mémoire de celles-ci.

Le graphique suivant donne l'occupation mémoire en fonction du nombre de paramètres entrées par seconde et le temps de création d'une image.

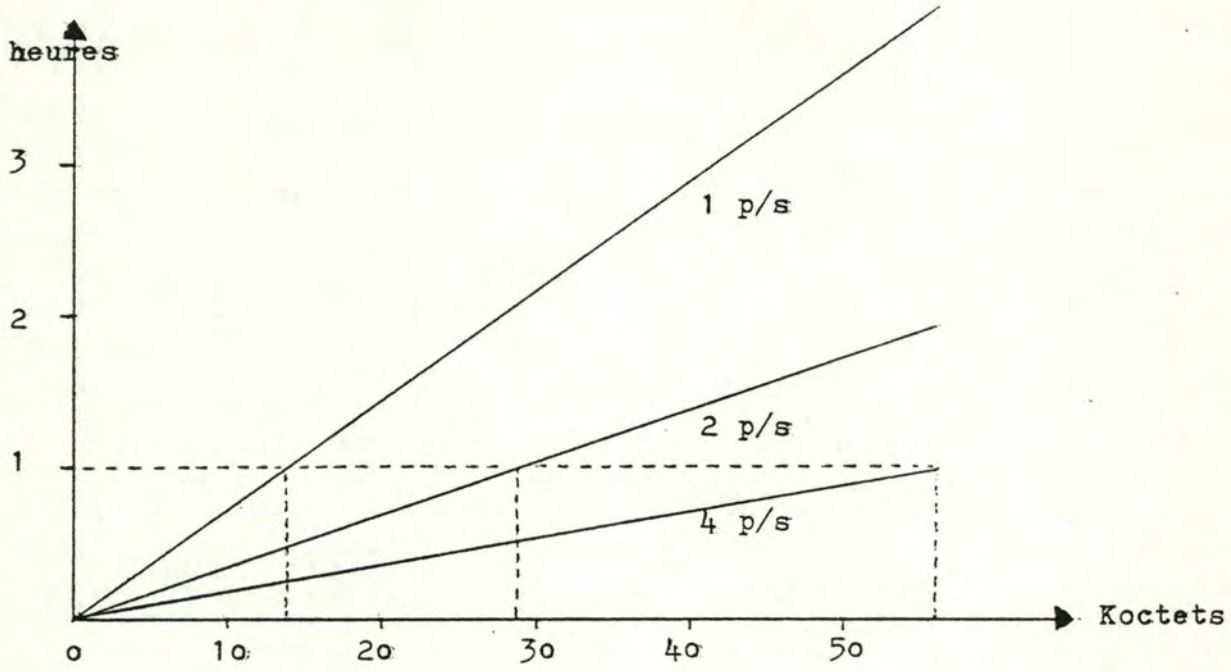


FIGURE 3.8 : OCCUPATION MEMOIRE DES IMAGES PICASSO

Après 1 heure de travail
à 2 p/s on occupe 30 Koctets
à 4 p/s " " 60 "
Les images stockées avaient en général une taille entre 20 à 40 Koctets. Dès lors, nous pensons qu'un rythme de travail de 2 paramètres/seconde est tout à fait acceptable.

Si nous définissons par T le taux de compression obtenue et que

$$T = \frac{\text{taille de l'image sous forme matricielle}}{\text{taille de l'image stockée}}$$

nous obtenons un taux de compression moyen de

$$T = \frac{768 \text{ Koctets}}{40 \text{ Koctets}} = 19,2$$

l'original prend donc 19 fois plus de place que l'image stockée sous forme de liste de coordonnées.

L'image étant ainsi stockée, il restait à trouver une procédé de codage adapté à cette structure de l'image.

3.2.3. Codage/Decodage d'images

Nous allons proposer deux méthodes de codage, dont seulement la première a été effectivement implementée. Les codages proposés appartiennent aux méthodes dites "programmées". Si elles sont moins générales et dépendent dans la majorité des cas de l'application, elles ont l'avantage de pouvoir être adaptées sur mesure aux caractéristiques du système utilisé.

Comme PICASSO est un produit isolé ne prétendant nullement à une mise en commun avec d'autres systèmes interactifs, une telle méthode de codage est tout à fait acceptable dans notre cas.

Jusqu'à présent, on n'a pas fait de différence entre stockage et codage, l'image étant toujours représenté sous forme matricielle auquelle le codage fut alors appliqué. Dans le système PICASSO, il y a moyen de voir les choses d'un autre angle. C'est ce que nous allons découvrir maintenant.

3.2.3.1. Méthode structurée

Les codages utilisés profitent de la structure interne de l'image telle qu'elle a été mémorisée. La première méthode est appelée "structurée" parce que chaque code reflète la structure hiérarchique des menus. Ceci permet une extension facile du code au dépens d'un taux de compression moins élevé comme nous allons le voir.

La structure arborescente des menus est indiquée sur la figure 3.9.

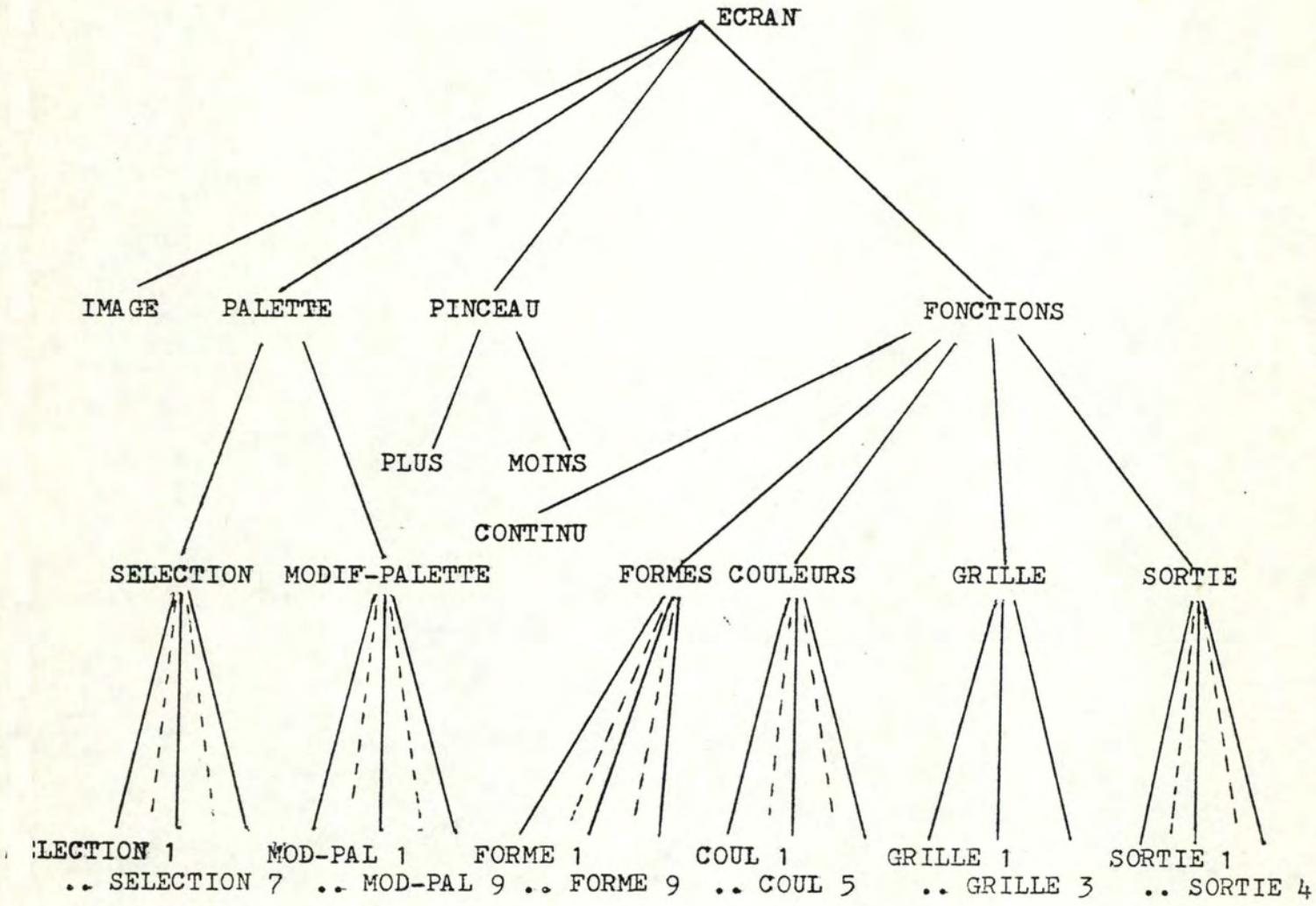


FIGURE 3.9 : STRUCTURE ARBORESCENTE DES MENUS DANS PICASSO

Le code

Dans la méthode de codage "structurée", nous avons choisi de ne coder que les feuilles des menus (commandes) et de reprendre les points de la zone image (paramètres) tels quels.

Le fait que la méthode est structurée implique nécessairement que chaque code isolée permet d'identifier n'importe quelle commande ET famille de commande.

Quelle est la taille nécessaire par code ?
Parler de code nous oblige évidemment de parler de compression et ainsi de taille du code, puisque la compression en dépend. Rappelons que la taille d'un paramètre ou d'un ordre est de 4 octets ou de 2 mots, chaque mot étant composé d'un octet des poids forts (octet-fort) et d'un octet des poids faibles (octet-faible).

- Pour permettre la distinction entre un paramètre et un code, il nous faut 1 bit.
- Les trois zones de menus peuvent être codées sur 2 bits.
- Le maximum de sous-noeuds d'un noeud de zone est 5 (menu des fonctions), 3 bits sont donc nécessaires.
  Remarque: On aurait pu distinguer entre les quatre familles et la fonction CONTINU en ne prenant que 2 bits; ceci aurait d'une part cassé la structure du code et d'autre part interdit une extension future.
- Le maximum de fonctions par famille est de 9, 4 bits suffisent donc pour les représenter toutes.

Il faut donc 8 bits pour représenter univoquement la famille du plus bas niveau plus 4 bits pour les fonctions, le code aura donc une longueur de 16 bits, soit 1 mot.

Nous avons convenu de coder les noeuds dans l'octet-fort et les fonctions dans l'octet-faible, le bit le plus fort du mot servant à identifier code ou paramètre.

Suivant ce qui vient d'être dit et la structure arborescente des menus, la structure d'un code dans le mot est la suivante:

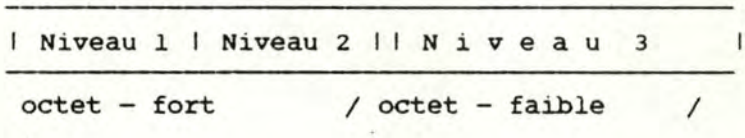


FIGURE 3.10 : REPRESENTATION INTERNE DU CODE "STRUCTURE"

Les codes utilisés sont les suivants:

- Au niveau 1, le seul qui soit indépendant des autres, les chiffres hexadécimaux 9, A et B codent la palette, le menu des fonctions et les pinceau respectivement.
- Au niveaux 2 et 3, dépendant du premier, le codage est énumératif.

Représentée sous forme arborescente, le code se présente comme suit:

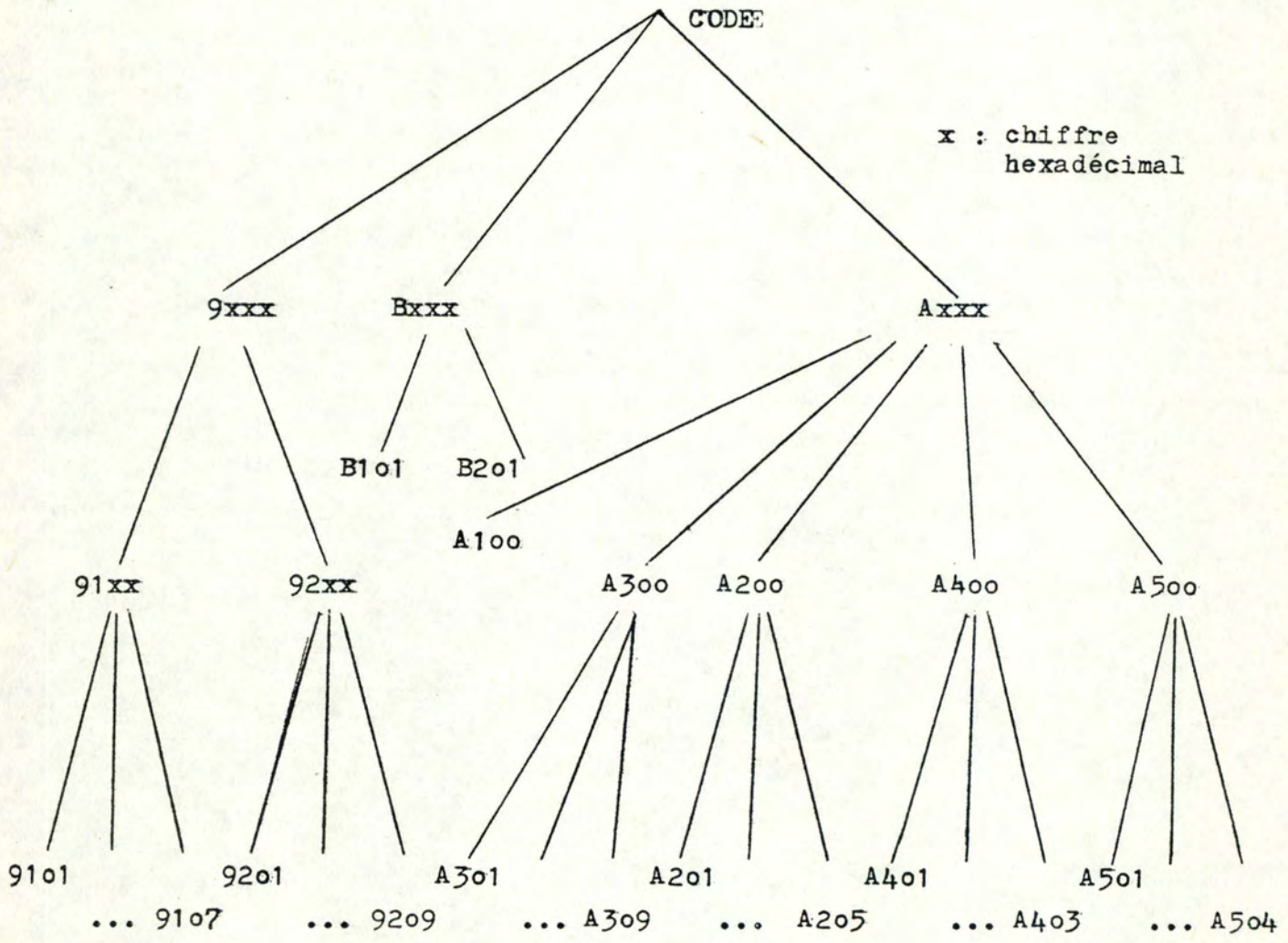


FIGURE 3.11 : STRUCTURE ARBORESCENTE DU CODE "STRUCTURE"

Evaluation du code

Soit I, M et D les tailles respectives de l'image entière, de la partie des menus et des paramètres.
Lors du stockage, nous avons

$$I = M + D$$

Nous avons déjà vu que le taux de compression obtenu lors du stockage était considérable, de l'ordre de 20. Ici, nous allons nous intéresser au taux de compression obtenu lors du codage par rapport à l'image stockée.

$$T = \frac{\text{taille stockée}}{\text{taille codée}} = \frac{M + D}{M' + D'}$$

avec $I' = M' + D'$ donnant la taille de l'image codée.
Dans le cas présent, $D' = D$ et $M' = M/2$.
T devient donc:

$$T = \frac{M + D}{\frac{M}{2} + D} = 2\,\frac{M + D}{M + 2D}$$

Posons $D = nM$, n étant un réel positif non nul
Le taux de compression devient:

$$T = 2\,\frac{1 + n}{1 + 2n}$$

et ne dépend plus que de n.
Deux cas sont à distinguer:

1. $0 < n < 1 \Rightarrow D < M$
   si $n \longrightarrow 0$, le taux devient maximal, puisque D, non codé, devient très petit.
   Le taux maximal obtenu est donc:

$$\lim_{n \longrightarrow 0} T = 2$$

   Le code permet donc au plus une réduction de moitié de la taille de l'image stockée.

2. n >= 1 => D >= M
   Si D = M, on a T = 4/3 le gain obtenu est donc d'un tiers
   si n —> ∞ , le taux de compression est minimal et égal à:

$$\lim_{n \to \infty} T = 1$$

ce qui revient à ne pas comprimer du tout.

Quelles conclusions en tirer ?
Le chiffre n peut varier beaucoup, selon l'objectif poursuivi par l'artiste, et les situations extrêmes sont rares, puisqu'une image est normalement composée de paramètres et d'ordres, la partie des paramètres étant souvent, pas toujours plus importante. Le taux de compression varie donc autour des 4/3, ce qui n'est pas beaucoup. Cependant, il faut remarquer que cette compression opère déjà sur une image comprimée et donc n'a pour but que d'améliorer le taux de compression obtenu lors de la phase de stockage.

Conscient du fait qu'il y a moyen de faire mieux, nous allons proposer une autre méthode de codage, non structurée cette fois-ci.

3.2.3.2. Méthode non structurée

La méthode précédente possède deux inconvénients majeurs:
1. Elle ne comprime pas du tout les paramètres, ce qui peut conduire à un taux de compression voisin de 1 dans certains cas.
2. Le code utilisé prend beaucoup de place, 1 mot mémoire, ce qui pourrait correspondre à plus de 32000 fonctions (eu égard au bit de différentiation entre paramètre et ordre), puisqu'en fin de compte, elles seules sont codées.

L'élimination de ces deux désavantages à conduit au code suivant.

Le code

Le code proposé est analogue au codage de Huffman, dans la mesure où il utilise une table de correspondance symbole-code, mais suppose tous les symboles (des menus) équiprobables et utilise donc une taille de code invariable. Celle-ci est de 1 octet, permettant un codage de 127 fonctions, donc le double de ce qui est actuellement disponible (il faut toujours réserver un bit de différentiation code-paramètre). Pour réduire la taille de la table de correspondance, on pourrait penser à un double codage, en codant d'abord selon la première méthode, puis selon la deuxième, une entrée dans la table de correspondance serait donc code2-code1.

Ce qui vient d'être dit ne concerne que l'un des désavantages de la première méthode de codage, mais ne permet toujours pas de réduire la taille des paramètres.

Analysons donc un paramètre: L'écran à une résolution de 512*512, image et menus compris. Il faut donc au plus 9 bits par abscisse (ou ordonnée), ce qui en fait 18 par paramètre. Cette taille tient dans 3 octets, ce qui permettrait de gagner 1 octet par paramètre.
Supposons le cas où un utilisateur travaille avec la fonction CONTINU. En général, dans ce cas, beaucoup de paramètres se succèdent dans l'image. Prenons un couple de paramètres, qui prend 4*2 = 8 octets ce qui correspond à 64 bits dans le cas normal. Or, 18 bits sont nécessaires pour un paramètre, donc 36 pour une paire. Soit 1 bit de différentiation paramètre-ordre et 1 bit de différentiation (paire de paramètres)-(paramètre isolée) + 36 bits effectifs, on en obtient 38, ce qui peut être codé dans 5 octets (40 bits). Cette solution permet de gagner encore 1 octet par paire de paramètres par rapport à celle où seuls les paramètres isolés sont codés (3*2 = 6 octets nécessaires). En plus, une extension est encore possible, les 2 bits restants permettant un quadruplement de la résolution actuelle.

Analysons dès à présent cette méthode de codage:

Evaluation

Dans le cas où nous ne considérons aucune distinction au niveau des paramètres, nous pouvons procéder de la même manière que pour la méthode structurée.

1. Si toutes les paramètres sont considérées égales au niveau du codage, l'image codée devient:

$$I'' = M'' + D''$$

avec $M'' = M/4$ et $D'' = (3D)/4$, le taux de compression T devient donc

$$T = \frac{M + D}{\frac{M}{4} + \frac{3D}{4}} = 4\,\frac{M + D}{M + 3D}$$

Si nous posons de nouveau $D = nM$, T devient

$$T = 4\,\frac{1 + n}{1 + 3n}$$

Selon les deux cas distingués précédemment, le taux est maximal pour n petit et minimal pour n grand. Ces taux sont:

$$\lim_{n \to 0} T = 4 \qquad \lim_{n \to \infty} T = 4/3$$

Si n = 1, T = 2. Les taux de compression obtenus dans les différentes situations sont donc bien meilleurs que ceux obtenus avec la méthode structurée.
Que se passe-t-il alors si on pousse la compression plus loin en distinguant entre paramètres isolés et paires de paramètres ?

2. Procédons d'une manière différente.
Soit I la taille de l'image,
M la taille des ordres,
D la taille des paramètres
m le nombre d'ordres donnés dans l'image,
d le nombre de paramètres,
d1 le nombre de paramètres isolées et
d2 le nombre de paires de paramètres.

On peut établir les relations suivantes:

$$I = M + D$$
$$M = 4m$$
$$D = 4d = 4(d1 + 2*d2)$$

Soient les rapports suivants

$$x = \frac{m}{d} \quad \Rightarrow m = xd$$

et

$$y = \frac{d2}{d1} \quad \Rightarrow d2 = y*d1$$

On peut donc reexprimer I à l'aide de ces rapports et de d1:

$$
\begin{aligned}
I &= 4(m + d) \\
&= 4(xd + d) \\
&= 4d(1 + x) \\
&= 4(d1 + 2*d2)(1 + x) \\
&= 4(d1 + 2y*d1)(1 + x) \\
&= 4*d1(1 + 2y)(1 + x)
\end{aligned}
$$

Pour calculer le taux de compression, il nous faut la taille de I". On a

$$
\begin{aligned}
I" = M" + D" \quad & \text{avec } M" = M/4 = 4m/4 = m \\
& \quad \text{et } D" = D1" + D2" \\
& \text{avec } D1" = 3*d1 \\
& \quad \text{et } D2" = 5*d2
\end{aligned}
$$

donc

$$I" = m + 3*d1 + 5*d2$$

et avec les relations définies ci-dessus, on obtient

$$
\begin{aligned}
I" &= m + 3*d1 + 5y*d1 \\
&= m + d1(3 + 5y) \\
&= xd + d1(3 + 5y) \\
&= x(d1 + 2*d2) + d1(3 + 5y) \\
&= x*d1(1 + 2y) + d1(3 + 5y) \\
&= d1[x(1 + 2y) + (3 + 5y)]
\end{aligned}
$$

Le taux de compression T devient:

$$T = 4 \frac{(1 + x)(1 + 2y)}{x(1 + 2y) + (3 + 5y)}$$

Il devient plus difficile d'évaluer le taux de compression, comme plus de variables interviennent. On peut cependant calculer les taux maximaux et minimaux.
Comme la compression est plus grande pour les ordres que pour les paramètres, le taux sera certainement maximal pour x grand. Dans ce cas, on aura:

$$\lim_{x \to \infty} T = 4$$

La compression jouant seulement sur les ordres, une compression sur les paires de paramètres n'a aucun effet et le taux obtenu est le même que pour la méthode sans différentiation des paramètres.
Il serait intéresser de connaître le comportement du taux si on fait varier y, rapport entre les paires de paramètres et les paramètres isolés.

$$\lim_{y \to \infty} T = 4 \lim_{y \to \infty} \frac{1 + x + 2y + 2xy}{x + 2xy + 3 + 5y}$$

$$= 4 \lim_{y \to \infty} \frac{2y(1 + x)}{y(2x + 5)}$$

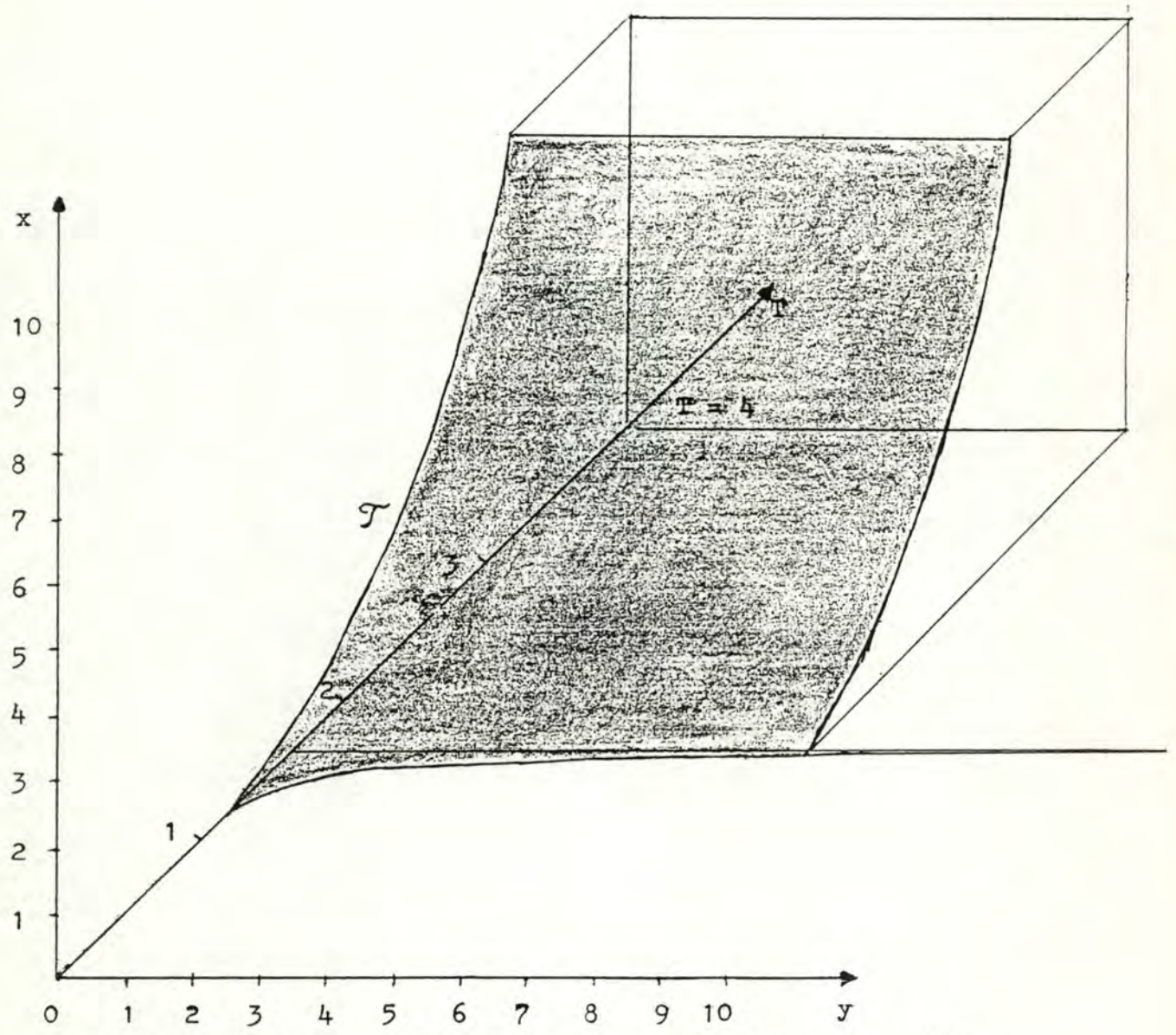
$$= 8 \frac{1 + x}{2x + 5}$$

Il est interessant de remarquer que cette fois-ci, le taux de compression dépend de x.
Pour x = 0, on obtient T = 8/5, cas où aucun l'image ne contient aucun ordre menu et que les parametres sont tous groupés en paires.
Pour $x \to \infty$, il faut calculer une limite qui nous donnera de nouveau 4 comme resultat.
Pour y = 0, on a

$$T = 4 \frac{1 + x}{3 + x}$$

et donc, si x = 0, T = 4/3, ce qui correspond au taux le plus bas qu'on peut obtenir, seulement realisable si l'image ne contient qu'un seul parametre, ce qui est fort rare !!
Si $x \to \infty$, le taux est évidemment 4.

Le graphique suivant illustre cette situation:



Les deux rapports x et y déterminent un plan de compression $\mathcal{T}$ qui se rapproche du plan de compression maximale T = 4 si x tend vers l'infini.

FIGURE 3.12 : VARIATION DU TAUX DE COMPRESSION DANS LA METHODE DE CODAGE NON STRUCTUREE (B)

On pourrait donc dire que le taux de compression est principalement influencé par le rapport x, mais que lors d'une variation de x, la vitesse de variation de T dépend de y.

Conclusion

Le problème du stockage/codage exposé, nous allons passer dès à présent à l'optimisation des images de PICASSO, dont le problème sera traité dans le chapitre suivant.

Chapitre 4 : Optimisation d'images

Introduction

Lorsqu'un peintre désire créer une image, deux situations peuvent se présenter: soit il sait ce qu'il veut faire, dans ce sens qu'il "voit" déjà l'image finale dans sa tête, soit il se laisse guider par son inspiration. Dans le premier cas, le problème principal pour le peintre est de choisir judicieusement le matériel nécessaire (pinceaux, couleurs) à l'élaboration de son oeuvre. Une connaissance parfaite du matériel est donc requise, les hésitations quant à l'utilisation de tel ou tel pinceau, telle ou telle couleur primant sur les hésitations ou erreurs commises pendant la phase effective de dessin.

Dans le deuxième cas, les hésitations sur les outils à utiliser seront probablement de moindre importance, l'artiste ayant plutôt tendance à essayer et puis d'en juger l'effet obtenu.

Dans les deux cas,on peut relever certaines erreurs de manipulations classiques:

1. L'utilisateur veut effectuer un traitement illicite, signalé par PICASSO, comme l'épaississement du pinceau, alors que son épaisseur est maximale
2. L'utilisateur peut donner plusieurs ordres "contradictoires", comme une saturation suivie d'une désaturation.
3. Il pourra faire appel à plusieurs fonctions, dont les effets seront "exclusifs", comme la superposition exacte de deux couleurs dans une même tâche.

Ces constatations nous amènent à faire plusieurs remarques:

1. Seul le premier point ci-dessus concerne des erreurs que PICASSO peut détecter et signaler.
2. Aux trois points énumérés ci-dessus correspondent deux niveaux d'optimisation différents, selon qu'il s'agit d'optimiser le <u>contexte</u> d'une fonction (c-à-d tout ce qui est nécessaire pour son exécution) ou <u>l'effet</u> d'une fonction.
   Remarque: Nous nous voyons contraint de passer immédiatement à l'exposé de notre travail, le problème de l'optimisation telle que nous le concevons n'étant pas repris dans la littérature (du moins celle qui était à notre disposition).

Seul le premier niveau sera traité maintenant alors que le deuxième niveau sera pris en considération dans la dernière partie.

Pour mieux poser le problème, nous allons d'abord essayer de définir le langage graphique utilisé dans PICASSO, en le présentant sous deux points de vue, tel qu'il est perçu par le système d'une part, et par l'optimiseur d'autre part. Ceci nous est possible en jouant sur la structure d'une image.

Une remarque s'impose néanmoins:

La syntaxe ne nous indiquera ni quoi, ni comment optimiser, l'interprétation du langage étant profondément différente du processus d'optimisation.

4.1. Syntaxe du langage graphique utilise

La syntaxe sera donnée à l'aide de la Backus/Naur/Form (B.N.F.). Un bref rappel en est donné ici.

4.1.1. La Backus/Naur/Form (BNF)

C'est un moyen de représentation d'une grammaire à contexte libre dont les symboles utilisés sont les suivants:

::=	-	indique une production de la grammaire
<NON-TERMINAL>	-	représentation d'un symbole non-terminal du langage
terminal	-	représentation d'un symbole terminal
{Symbole}	-	0 à n occurences de Symbole
Symb-1 | Symb-2	-	Symb-1 ou (exclusif) Symb-2
<VIDE>	-	Non terminal spécial, symbole vide

Pour la commodité de l'emploi, nous avons en plus utilisé des symboles de regroupement, à savoir "[" et "]". On a donc :
<A> [<B> | <C>] = <A> <B> | <A> <C>
De même, le symbole ".." est utilisé pour indiquer un intervalle, pour ne pas devoir donner une liste exhaustive de symboles analogiques. Ainsi :
<A> | <B> | <C> | <D> = <A> | .. | <D>

4.1.2. Syntaxe du langage graphique vue par PICASSO

La syntaxe que voici indique en même temps une structure de l'image, telle qu'elle est perçue par l'utilisateur.

```
<IMAGE>            ::= {<PARAMETRE>} {<PAS D'IMAGE>}
<PARAMETRE>        ::= <CONTEXTE> coordonnée-image
<CONTEXTE>         ::= <VIDE> |
                       <PINCEAU> <CONTEXTE> |
                       <PALETTE> <CONTEXTE>
<PINCEAU>          ::= plus |
                       moins
<PALETTE>          ::= <BLOC-COULEUR> |
                       <ACTION-COULEUR>
<BLOC-COULEUR>     ::= pavé-1 | .. | pavé-7
<ACTION-COULEUR>   ::= clair | vif | blanc | noir |
                       plus | moins |
                       voisines | retour | défiler
```

```
<PAS D'IMAGE>     ::= <CONTEXTE>
                      [continu <PARAMETRE> {<PARAMETRE>} |
                       formes <FORME>                    |
                       couleurs <COULEUR>                |
                       grille <GRILLE>                   |
                       sortie <SORTIE> ]

<FORME>           ::= <CONTEXTE>
                      [<FORME-2-PARAM> <DEUX-PARAM>
                       {<DEUX-PARAM>}                   |
                       <FORME-3-PARAM> <TROIS-PARAM>
                       {<TROIS-PARAM>}]

<FORME-2-PARAM> ::= segment          |
                    ligne-brisee     |
                    triangle         |
                    carre            |
                    parallèlogramme  |
                    losange          |
                    rectangle        |
                    cercle
<FORME-3-PARAM> ::= ellipse

<COULEUR>         ::= <CONTEXTE>
                      [remplir <UN-PARAM>              |
                       mèlange <UN-PARAM>              |
                       couleur-sur-image <UN-PARAM>    |
                       fond <UN-PARAM>                 |
                       effacer {<PARAMETRE>}]

<GRILLE>          ::= <CONTEXTE>
                      [plus {plus}        |
                       moins {moins}      |
                       enlever ]
<SORTIE>          ::= <CONTEXTE>
                      [photo   |
                       reset   |
                       texte ]
<UN-PARAM>        ::= <PARAMETRE> {<[PARAMETRE>}
<DEUX-PARAM>      ::= <PARAMETRE> <PARAMETRE>
<TROIS-PARAM>     ::= <DEUX-PARAM> <PARAMETRE>
```

Ce qui nous paraît essentiel à ce stade est le fait que l'image est présentée comme un flot d'ordres et de paramètres, divisé logiquement en pas d'image qui correspondent aux différentes fonctions utilisées plus leur contexte de modification. C'est donc la fonction qui détermine le début d'un pas d'image.
Remarquons finalement qu'aucun contrôle de validité n'est spécifié et que nous n'avons inclus aucune contrainte dans la syntaxe, pour la simple raison que les erreurs au sens stricte du terme n'existent pas dans PICASSO, le processus de création ne pouvant être influencé ou interrompu par une erreur.

4.1.3. Syntaxe du langage graphique telle que vu par l'optimiseur

La structure de l'image sera modifiée de telle manière qu'une utilisation simple de la BNF ne permet plus de la représenter intégralement. Nous avons donc ajouté des contraintes d'existence et de cardinalité qui seront explicitées au moment nécessaire.

```
<IMAGE>          ::= {<PAS INITIAL>} {<PAS D'IMAGE>}
<PAS INITIAL>    ::= <CONTEXTE> coordonnée-image
<CONTEXTE>       ::= <VIDE> |
                     <PINCEAU> |
                     <PALETTE>
<PINCEAU>        ::= plus {plus} (2) |
                     moins {moins} (2)
<PALETTE>        ::= <SELECTION> |
                     <MODIF-PAL>
<SELECTION>      ::= pavé-1 |
                     pavé-2 |
                     ..     |
                     pavé-7

<MODIF-PAL>      ::= clair                    |
                     vif                      |
                     blanc                    |
                     noir                     |
                     defiler                  |
                     plus {plus} (2)          |
                     moins {moins} (2)        |
                     voisines {voisines} (2)  |
                     retour {retour}

<PAS D'IMAGE>    ::= <CHAMP-FORMES> |
                     <CHAMP-COULEURS> |
<CHAMP-FORMES>   ::= <FAMILLE-FORMES> <ACTION-FORMES> (1)

<FAMILLE-FORMES> ::= <VIDE> |
                     <CONTEXTE-FAM> formes
<CONTEXTE-FAM>   ::= <VIDE> |
                     <CONTEXTE> <CONTEXTE-FAM> |
                     <CONTEXTE-GRILLE> <CONTEXTE-FAM>
<CONTEXTE-GRILLE>::= <VIDE> |
                     [grille <MODIF-GRILLE> |
                      <MODIF-GRILLE> ]
<MODIF-GRILLE>   ::= plus {plus} (2)  |
                     moins {moins} (2)|
                     enlève

<ACTION-FORME>   ::= {<CONTEXTE-ACTION>}
                     [<FORME-2-PARAM> <CHAMP-2-PARAM> |
                      <FORME-3-PARAM> <CHAMP-3-PARAM> ]
```

```
<CONTEXTE-ACTION>::= <VIDE> |
                     <CONTEXTE> <CONTEXTE-ACTION> |
                     <MODIF-GRILLE> <CONTEXTE-ACTION>
<FORME-2-PARAM>  ::= <VIDE> |
                     segment |
                     ligne brisée |
                     triangle |
                     carré |
                     rectangle |
                     losange |
                     parallélogramme |
                     cercle

<FORME-3-PARAM>  ::= <VIDE> |
                     ellipse
<CHAMP-2-PARAM>  ::= <PARAMETRE> <PARAMETRE>
<CHAMP-3-PARAM>  ::= <PARAMETRE> <CHAMP-2-PARAM>
<PARAMETRE>      ::= {<CONTEXTE-ACTION>} coordonnée-image

<CHAMP-COULEUR>  ::= <FAM-COULEURS> <ACTION-COULEURS> (1)
<FAM-COULEURS>   ::= <VIDE> |
                     <CONTEXTE-FAM> couleurs
<ACTION-COULEURS>::= {<CONTEXTE-ACTION>}
                      [<COULEUR-1-PARAM> <PARAMETRE> |
                       <EFFACER> {<PARAMETRE>}

<COULEUR-1-PARAM>::= <VIDE> |
                     fond |
                     couleur-sur-image |
                     mélange |
                     remplir
<EFFACER>        ::= <VIDE> |
                     effacer
```

(1) : contrainte d'existence implicite : de par la structure des menus, une commande ne peut être donnée que si sa famille a été sélectionnée, dans le pas d'image actuel ou l'un des précédents.
(2) : contrainte de cardinalité : on ne peut faire des modifications qu'entre certaines bornes dépendantes de la commande.

La grille ne fait plus partie du pas d'imagec comme elle ne sert ps à peindre, mais seulement à ajuster les coordonnées lors de l'utilisation des formes géométriques et la famille SORTIE ne fait plus partie de l'image, son usage étant limité à la création.
Cette structure sera à la base de l'optimisation

4.2. Principes d'optimisation

Sans entrer dans les détails de l'optimisation, nous allons essayer de dégager maintenant quelques principes qui aident à déterminer ce qu'il faudra optimiser.

1. Principe d'independance entre zones

   Ce principe découle directement de la structure arborescente des menus et du fait que les trois zones de menus représentent des familles inclusives. En plus, les commandes sont fondamentalement différentes des paramètres puisqu'elles spécifient le mode d'utilisation de ceux-ci.

2. Principe d'identité

   Découle du deuxième type de redondance défini au point 3.1.4.

3. Principe de quasi-décomposabilité entre pas d'images

   Ce principe stipule qu'il est possible, dans la plupart des cas, de décomposer l';image en petites unités appelées pas d'image, qui importent et exportent de l'information concernant l'état de l'image.

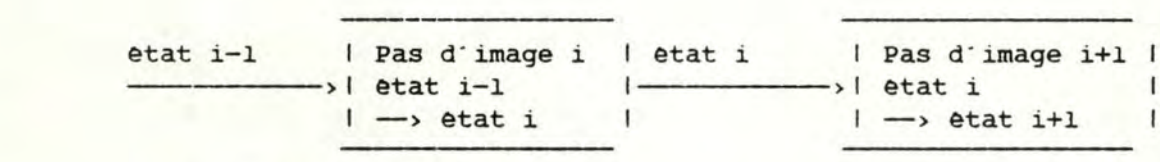


FIGURE 3.13 : ECHANGE D'INFORMATIONS ENTRE PAS D'IMAGES

4. Principe d'exclusion

   Ce principe est déduit de la structure des menus de nouveau, en se basant sur le fait que tous les noeuds et feuilles d'une zone sont exclusifs.

5. Principe du dernier accepté

   Ce principe complète le précédent puisqu'il permet de déterminer quelle famille ou commande il faut accepter lorsqu'il y a exclusion.

6. Principe du chemin optimal

   Ce dernier principe fait appel à la notion de chemin d'accès unidirectionnel. Le principe s'applique toujours à une suite d'ordres définissant un chemin et peut ainsi ne pas être le chemin minimal existant. Par chemin minimal, nous endendrons le ou les chemin(s) nécessitant le moins de commandes pour atteindre une cible à partir d'une origine.

Etudions maintenant les optimisations à faire en nous basant sur le principe de décomposabilité entre zones.

4.3. Le menu des fonctions

On peut distinguer les familles GRILLE et SORTIE des deux autres sur plusieurs points de vue: d'abord elles ne nécessitent toutes les deux pas de paramètres, ensuite ce sont les seules familles qui ne contribuent pas à l'image, leur effets n'étant pas visibles; ce sont des familles d'aide. En ce qui concerne la famille SORTIE, elle est même dénouée de sens puisqu'aucune fonction de sortie ne peut être mémorisée, afin d'éviter un bouclage ou une réexécution d'une telle fonction. La grille peut être remplacé par un traitement d'ajustement automatique des coordonnées.

Les noeuds et les feuilles peuvent être classés en deux catégories : ceux (celles) dont la portée est ponctuelle et ceux (celles) dont la portée est unidimensionnelle.

Les noeuds ou feuilles de la première catégorie sont ceux (celles) dont la sélection répétive désigne toujours le même point dans l'espace, alors qu'une sélection répétitive d'un élément de la deuxième catégorie a pour effet un mouvement dans l'espace. Tous les noeuds appartiennent à la première catégorie. Parmi les feuilles, appartiennent à cette catégorie celles auxquelles correspondent des commandes de sélection, alors qu'aux feuilles de la deuxième catégorie correspondent les commandes de modification. Les principes d'identité, d'exclusion et du dernier accepté s'appliquent aux familles et aux commandes de sélection, le principe du chemin optimal s'appliquant aux seules commandes de modification.

4.4. Le pinceau

La zone pinceau ne concerne que deux commandes de la deuxième catégorie: il est donc possible de trouver un chemin optimal. La portée des commandes est unidimensionnelle et le cheminn optimal unidirectionnel, le seul chemin acceptable est donc celui d'une augmentation ou d'une diminution de l'épaisseur pendant un pas d'image, selon le principe de quasi-décomposabilité.

4.5. La palette

Selon ce qui vient d'être dit, on pourrait appliquer le même raisonnement à la palette.

Or, l'optimisation de la palette peut être vue selon un autre point de vue, on se basant toujours sur les principes d'optimisation définis. Pour ce faire, on se base sur la structure même des modèles des couleurs utilisés dans PICASSO. Comme nous l'avons déjà vu lors de la présentation des modèles, la teinte est indépendante des deux autres facteurs qui sont l'intensité et la saturation.

L'optimisation de la palette se fera donc sur deux niveaux différents: au premier niveau, on classera les commandes selon qu'elles concernent la teinte ou l'un quelconque des autres critères; au deuxième niveau seulement se fera alors la classification exposée plus haut.

4.5.1. Intensité/Saturation/Blanc/Noir

Cette optimisation concerne les deux commandes de modification NOIR et BLANC ainsi que les quatre commandes de sélection CLAIR, VIF, BLANC et NOIR. Le chemin optimal des commandes PLUS et MOINS répond aux mêmes critères que celui de la grille.

4.5.2. Teinte

Le cas de la teinte est plus compliqué, rien que par le nombre de commandes impliquées: quatre commandes de modification et huit de sélection. Les mêmes principes que ceux appliqués au point précédent peuvent donc être appliqués. Or, comme trois groupes de commandes différents interviennent (sélection d'une teinte parmi celles visualisées, affinement ou grossissement du voisinage de la teinte courante et parcours du cercle des teintes), on pourrait se demander s'il n'y a pas moyen de trouver un chemin d'accès optimal à une teinte, indépendamment des chemins optimaux définis pour les commande de modification.

Deux constatations permettent de mieux comprendre le chemin proposé:

1. Le défilement séquentiel n'influence pas la teinte courante, il ne fait que la décaler par rapport à sa position actuelle.
2. Les commandes d'avoisinage centrent la teinte sélectionnée dans la palette affichée, si possible.

Nous utiliserons de nouveau la BNF pour exposer la solution proposée:

```
<CHEMIN OPTIMAL> ::= <ACCES-TEINTE> |
                     <CHOIX-TEINTE> <ACCES-TEINTE> |
                     <ACCES-TEINTE> <AVOISINAGE>
<ACCES-TEINTE> ::= <VIDE> |
                   <AVOISINAGE> <CHOIX-TEINTE> <ACCES-TEINTE>


<AVOISINAGE> ::= <GROSSIR> |
                 <AFFINER>
<GROSSIR> ::= retour {retour} (1)
<AFFINER> ::= voisines {voisines} (1)
<CHOIX-TEINTE> ::= <DEFILER> <SELECTION>


<DEFILER> (2) ::= {<DEFILER-GAUCHE>} |
                  {<DEFILER-DROITE>}
<DEFILER-GAUCHE> ::= plus {plus} (1)
<DEFILER-DROITE> ::= moins {moins} (1)
<SELECTION> ::= pavé-1 |
                ..     |
                pavé-7
```

(1) : contrainte de cardinalité : nombre d'occurences dépend de la commande
(2) : on suppose dans ce cas que la commande de sélection DEFILER a été choisie.

Illustrons cette grammaire par un exemple :
Les teintes seront représentées par leur emplacement sur le cercle des teintes en degrés.
Un chemin possible serait le suivant:

```
[VOISINES-VOISINES] [PLUS-PLUS-PAVE7] [MOINS-PAVE2] [RETOUR]
          1                 2               3          4

[VOISINES] [PLUS-PAVE1] [PAVE7] [RETOUR] [PAVE7]
     5         6          7       8        9
```

Les commandes sont groupées selon <CHOIX-TEINTE> et <AVOISINAGE>.
Selon la grammaire, 2 et 3 ne peuvent se suivre, on applique donc les principes du chemin optimal et du dernier accepté pour obtenir [PLUS-PAVE2]. Idem pour 4 et 5 qui s'éliminent par le principe du chemin optimal. 6 et 7 vont être transformés en [PLUS-PAVE7], etc.
En fin de compte, le chemi optimal est le suivant:

```
[VOISINES-VOISINES] [PAVE7] [RETOUR] [PAVE7]
          1           2        3       4
```

En supposant les teintes representees par leur emplacement sur le cercle des teintes (en degrés), la figure suivante montre le chemin parcouru:

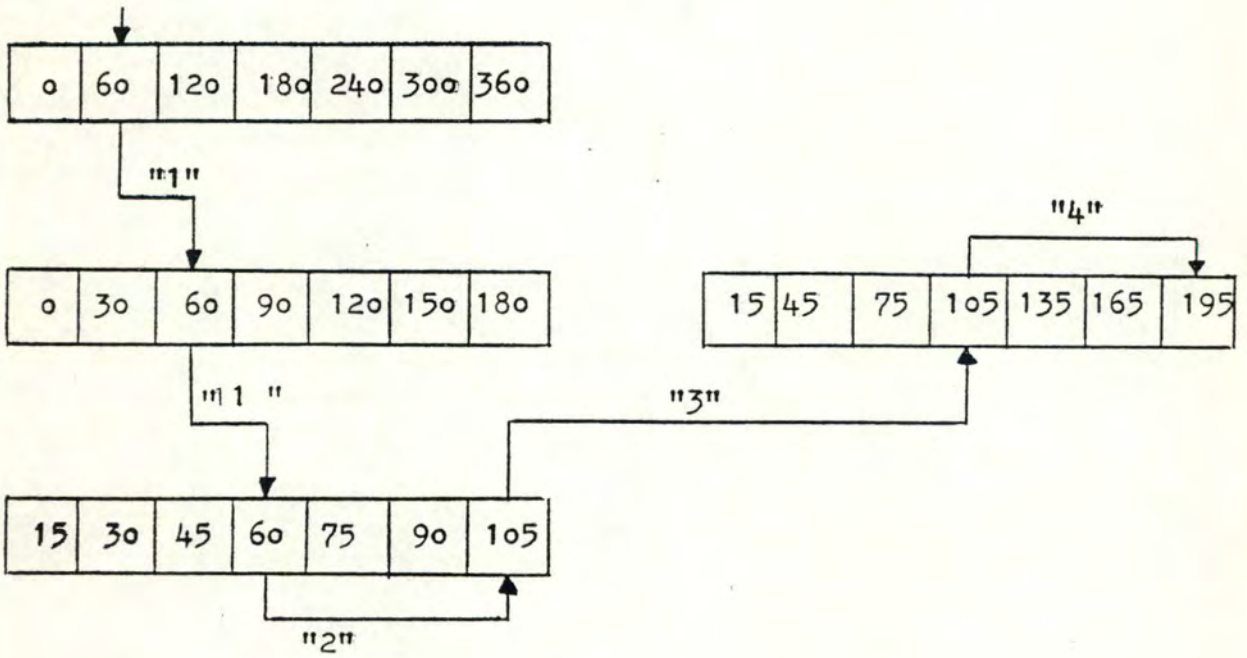


FIGURE 4.1 : EXEMPLE DE CHEMIN D'ACCES
OPTIMAL A UNE TEINTE

Malheureusement, ce chemin optimal n'est pas encore minimal. En effet, pour aboutir à une teinte particulière, on ne tient compte que des commandes données par l'utilisateur qui ne sait normalement pas comment acceder le plus rapidement à une teinte voulue. Cette situation peut être très gênante puisque le rapport entre le nombre de commandes du chemin minimal et le nombre de commandes du chemin optimal peut devenir très petit.
Il se peut même que l'utilisateur boucle dans sa recherche sans que cette "erreur" ne soit détectée.

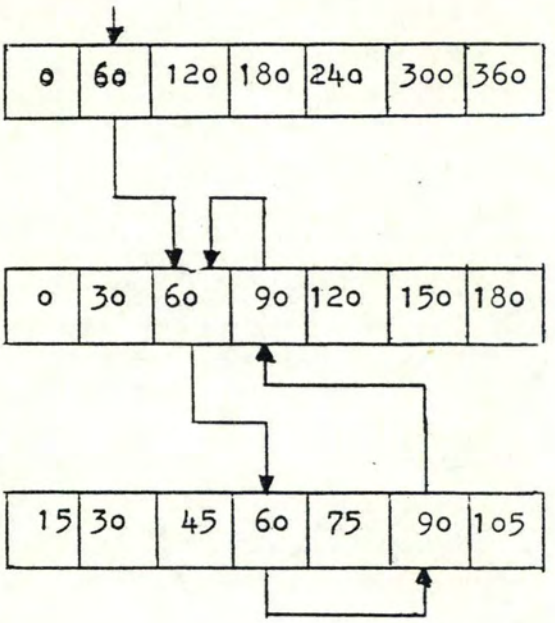


FIGURE 4.2 : BOUCLE DANS UN CHEMIN OPTIMAL

D'après la grammaire donnée, cette situation est tout à fait acceptable, la notion de teine particulière lui faisant defaut. Nous donnerons dans les extensions possibles à l'optimisation, quelques idées relatives à l'obtention d'un chemin minimal.

Conclusion

Dans le présent chapitre, nous venons de présenter la fonction d'optimisation de PICASSO. Nous croyons qu'elle est assez originale, puisqu'à notre connaissance, aucun autre système ne l'offre, et en même temps utile, la recherche dans les couleurs ou l'affichage de la grille prenant beaucoup de temps.

A part les fonctions de stockage/codage et d'optimisation, d'autres fonctions ont été jugées intéressantes pour faire partie intégrante de PICASSO, devenant seulement réalisables à partir du moment où le système disposait d'une possibilité de mémorisation. Ce sont ces fonctions qui feront l'objet du prochain chapitre.

Chapitre 5 : FONCTIONS SUPPLEMENTAIRES

Introduction

Dans le cadre de l'ajout d'une fonction de stockage au logiciel PICASSO, il est à noter que d'autres fonctions pouvant intéresser les utilisateurs ont été réalisées.

Nous allons les présenter ici,en reportant toutefois une étude plus profonde sur des fonctions à ajouter en général à la dernière partie du présent exposé, traitant d'une éventuelle évolution de PICASSO.

5.1. Mise-à-jour d'images

Les fonctions décrites ci-dessous sont dites "globales" car elles traitent des images entières, par opposition aux fonctions de manipulation "locales" pouvant opérer sur des éléments d'une image (déplacement d'un objet dans l'image p.ex). Pour ce dernier genre de fonctions, il faut disposer d'une primitive de détourage permettant d'isoler un objet du reste de l'image. Ce cas sera traité dans la troisième partie.

Pourquoi ces fonctions ont-elles été réalisées ?

Comme nous l'avons déjà mentionné, le point de départ de notre travail était le besoin de mémorisation d'une image créée sur PICASSO, exprimé par la plupart des utilisateurs. Or, à cette nécessité de stockage est directement lié celui de mise-à-jour d'une imgage, le fait de disposer de la seule fonction de stockage n'étant pas intéressant.

Quelles fonctions réaliser ?

Avant de pouvoir mettre à jour quoi que ce soit, voire tirer un minimum de profit du stockage, il est évidemment prioritaire de disposer d'un moyen d'affichage de l'image mémorisée. Cette fonction était donc la première à mettre en oeuvre. La m-à-j devient donct possible dès ce moment.

Qui dit mise-à-jour dit modification; ceci nous amène à trois problèmes différents:
veut-on modifier l'image après, avant ou même pendant l'affichage de l'image stockée?

Dans le premier cas, il s'agit d'abord d'afficher l'image et ensuite de donner la main à l'utilisateur pour lui permettre d'effectuer des modifications.

Dans le deuxième cas, le problème qui se pose est celui de la superposition de l'image stockée à une image déjà présente sur l'écran.

Le dernier cas est le plus compliqué car il faut permettre à l'utilisateur d'interrompre le processus d'affichage, de modifier l'image et puis de demander une continuation de l'affichage interrompu. On en parlera dans le cadre des extensions à MACCOI dans le chapitre 7.

Pour compléter ces fonctions de mise-à-jour, il nous a paru élémentaire de permettre, en plus des modifications, de détruire ou de renommer une image.

5.1.1. Détruire

Cette fonction permet à l'utilisateur de rayer de manière définitive une image de sa bibliothèque.

5.1.2. Renommer

Comme des modifications apportées à une image peuvent en détourner la signification originale, cette fonction permet de renommer une image avec un nom mieux adapté à son contenu.

5.1.3. Afficher

Cette fonction de PICASSO permet de visualiser une image mémorisée en simulant le moyen d'entrée. Comme nous venons de la voir dans la partie précédente, PICASSO interprète les ordres reçus un à un sous forme de coordonnées. A partir du moment où la lecture de la coordonnée est faite, le décodeur d'ordres de PICASSO n'a plus aucun moyen de déterminer la provenance de cet ordre. L'objectif de la fonction d'affichage est donc d'injecter les coordonnées dans PICASSO en appelant après chaque coordonnée le décodeur qui lui, invoque les modules d'exécution. La fonction d'affichage, si elle n'est pas interrompu pendant son exécution, donne un effet d'animation pouvant être exploité à certaines fins, les temps morts dus aux réflexions de l'utilisateur étant supprimés.
De cette manière, la dynamique de l'affichage est seulement liée à la vitesse de traitement.

L'image se trouvant maintenant sur l'écran préalablement vide, le système rend la main à l'utilisateur lui permettant ainsi de mettre à jour son image.

5.1.4. Superposer

Nous utilisons ici le terme de superposition, plus agréable que celui de fonction de mise-à-jour pré-affichage. Cette fonction permet donc en fait de créer une image nouvelle, puis de lui en superposer une autre.

La différence essentielle avec la fonction précédente est qu'il faut garder l'image éventuellement présente sur l'écran. La superposition se réduit dès lors à un affichage sans destruction préalable. Ceci pose un problème puisque la construction de l'image se fait toujours à partir de l'état initial, alors que la machine se trouve dans un état indéterminable avant la construction automatique. Ceci ne concerne évidemment que les zones menus de l'écran. Une solution consiste à modifier la structure de l'image de la manière suivante:

```
<IMAGE> ::= <VIDE> |
            début {<PARAMETRES>} {<PAS D'IMAGE>} <IMAGE>
```

5.2. Fonctions particulières

5.2.1. Effacement automatique

Lors de certaines fausses manipulations sur PICASSO, une fonction de retour en ARRIERE avec effacement automatique de l'erreur s'avère être tres intéressante. En effet, lorsqu'on utilise la fonction REMPLIR, il se peut que le contour de la tâche a colorier ne soit pas fermé et que tout l'écran commence à se remplir avec la couleur courante. Ceci est en général un effet assez indésirable; c'est pourquoi PICASSO offrait déjà un moyen d'interruption de ce processus. Or, si les effets sont amoindris, ils ne sont pourtant pas éliminés, l'écran étant partiellement rempli avec une couleur non voulue. Avant l'existence d'une fonction de stockage, il était impossible de résoudre le problème autrement, PICASSO ne disposant pas d'historique de l'image; dès lors, il lui était impossible de connaître l'origine de l'erreur.

La fonction de stockage faisant partie intégrante du système, il est devenu possible de penser à une solution plus complète de ce problème. Cette solution se base évidemment sur la structure interne de l'image, structure qui n'était pas explicite dans PICASSO. Comme le lecteur pourrait être intéressé au fonctionnement de la fonction, la structure détaillée est donné ici (pour ne pas redire certaines choses, le lecteur est prié de se référencer au point 4.1.2):

```
<IMAGE>          ::= <..> {<PAS D'IMAGE>}0->m
<PAS D'IMAGE>    ::= <..> <CHAMP-PARAM>
                      (le champ dépend de la fonction)
<CHAMP-PARAM>    ::= {<PARAMETRE>}0->n |
                     {<2-PARAM>}0->n {<PARAMETRE>}0->1 |
                     {<3-PARAM>}0->n {<PARAMETRE>}0->2
```

On dira que le Pas d'Image n'est pas complet si n = 0, et qu'il est complet si n > 0.
La fonction procède en quatre étapes:

1. recherche du dernier pas d'image complet
2. recherche des derniers paramètres associés à ce pas d'image
   On considère ici la structure de l'image du point 4.1.3
3. éliminer ce(s) paramètre(s) de la structure de l'image
   Cette étape conduit à un réarrangement de la structure interne de l'image.
4. réafficher l'image complete, PICASSO ne permettant pas les modifications sur l'image affichée.

Ainsi, la structure du dernier pas d'image complet est la suivante après l'exécution de la fonction ARRIERE (pour n > 0):

```
<PAS D'IMAGE>    ::= <..> <CHAMP-PARAM>
<CHAMP-PARAM>    ::= {<PARAMETRE>}0->n-1 |
                     {<2-PARAM>}0->n-1 |
                     {<3-PARAM>}0->n-1
```

Conclusion

Dans la partie précédente, on vient de faire connaissance avec les extensions apportées au système PICASSO. Celui-ci peut être considéré dès à présent comme un système complet de C.A.O., puisque la fonction de structuration des données vient d'être réalisée.

Cette extension nous oblige à resituer PICASSO par rapport aux champs définis dans la première partie. La reconnaissance de formes et la compréhension d'images ne sont toujours pas présentes, mais le champ du traitement d'images tel qu'il a été défini a gagné de l'importance par l'adjonction de la fonction de gestion des données.

La reconnaissance de formes ou la compréhension d'images peuvent intervenir à un autre niveau, celui de l'analyse des images créées à l'aide de PICASSO, en analysant les images quant à leur contenu et en créant des descripteurs. Ces analyses peuvent être très diverses, puisque tout le pricessus de création est inhérent à la structure de l'image.

Finalement, on peut dire que la fonction de stockage donne un net avantage à PICASSO par rapport aux autres palettes du même niveau, qui ne disposent en général pas d'une telle fonction.

Pourtant, le système possède des limitations sérieuses aussi bien au niveau du matériel qu'au niveau du logiciel. C'est la raison pour laquelle une troisème partie propose des extensions à PICASSO aux deux niveaux, ainsi qu'à MACCOI au niveau logiciel exclusivement.

Partie 3 : EVOLUTION POSSIBLE DE PICASSO

Introduction

Ayant décrit dans les parties précédentes ce qu'il y a, ce qui est fait, tournons-nous dès à présent vers le futur pour découvrir quelques possibilités d'extension du projet PICASSO, tel qu'il est décrit dans [CHARROUF 84] d'une part, et des fonctions ajoutées par après, d'autre part. Ces deux cas seront traités séparément dans la suite.

Chapitre 6 : DIFFERENCES ENTRE PICASSO ET AUTRES PALETTES

Introduction

Dans le chapitre présent, on se basera essentiellement sur [CEPE 84] pour parvenir à notre but, c-à-d la description, ou simplement l'énumération dans quelques cas, d'un certain nombre de caractéristiques qui distinguent un logiciel de CAO artistique du bas de la gamme, tel que PICASSO, d'une palette du haut de gamme telle que QUANTEL, AURORA ou VIDIFONT. La première différence (peut être la plus importante pour l'utilisateur) est certainement le prix, celui de PICASSO avoisinant les 100.000 FF alors qu'il faut débourser entre dix et quinze fois plus pour une palette de haute gamme.

Une palette telle que QUANTEL, considérée comme étant la meilleure actuellement sur la marché, doit nécessairement offrir plus à ses utilisateurs, du point de vue matériel et du point de vue logiciel, pour justifier cette différence de prix. Si on peut admettre que le coût de mise en oeuvre d'une pièce de matériel est facilement estimable, les chiffres étant fournis par les constructeurs, il paraît beaucoup plus difficile d'estimer la valeur d'une fonction, étant donné que l'impact de celle-ci sur l'utilisateur n'est pas toujours le même, les différentes classes d'utilisateurs professionnels visées ayant des besoins différents et consentant dès lors à payer différemment les fonctions qu'on leur propose.

Nous essayerons pour cette raison de donner d'abord les caractéristiques matérielles intéressantes à mettre en oeuvre et ensuite les fonctions les plus demandées par toutes les classes d'utilisateurs.

Mentionnons encore le développement d'une nouvelle palette, au laboratoire d'informatique de l'université de Lille-1, visant à concurrencer les palettes de haute gamme et dont les caractéristiques générales rejoindront celles qu'on exposera par la suite, les détails n'étant soit pas encore connues, soit pas divilgués.

6.1. Au niveau du matériel

Le matériel intéressant à ajouter peut être incorporé à plusieurs endroits: en entrée, en sortie ou au traitement.

6.1.1. Moyens d'entrée

Le seul moyen d'entrée est actuellement le photostyle, ressemblant à un stylo ou à un pinceau et dès lors très facile à utiliser, mais qui présente l'inconvénient de forcer l'utilisateur à se trouver très près de l'écran et de se fatiguer plus rapidement les yeux ou le bras qu'avec un autre moyen d'entrée .

Deux autres moyens d'entrée peuvent être utilisés; ce sont la souris ou la tablette avec stylet.
L'utilisation de la souris nous paraît être particulièrement intéressante, comme elle permet des mouvements très petits sur l'écran et donc très précis, son propre déplacement pouvant être un multiple de la largeur de l'écran.

Un autre moyen d'entrée, différent des deux autres, quant à son utilisation, est la caméra vidéo. Elle sert surtout à la lecture d'images photographiques, digitalisées par après, auxquelles on peut appliquer des traitements divers tels que l'affichage d'un nombre restreint de couleurs.

6.1.2. Moyens de sortie

La présence d'une fonction de stockage, entraine nécessairement la présence d'un moyen de stockage, tel que un disque dur.

D'autres moyens de sortie, définitive cette fois-ci, peuvent être:

- une imprimante couleur
- une photo instantanée (polaroid)
- une bande vidéo
- un magnétoscope

On peut encore citer l'utilisation d'un deuxième écran vidéo, celui consacré au dessin ayant une meilleure résolution que celui actuellement utilisé.

6.1.3. Autres

Parmi le matériel non utilisé pour les E/S, on peut citer l'utilisation d'un processeur graphique plus performant (du point de vue de la manipulation interactive d'images) et plus rapide (lors de l'affichage d'une image ou du remplissage d'une tâche) et une mémoire vive plus grande, pouvant stocker des images intermédiaires.

Si les ajouts possibles paraissent peu nombreux, disons simplement que le rapport de prix entre l'écran utilisé dans PICASSO et le dernier produit de TEKTRONIX est de 1 à 10. S'ajoutent à cela les prix des disques durs et du matériel vidéo, on peut aisément concevoir le prix élevé de palettes disposant de tous ces outils.

6.2. Au niveau du logiciel

Si on analyse les différentes palettes présentes sur le marché informatique, on se rend compte bien vite que les différences qui les caractérisent ressortent plus des aspects logiciels que matériels. Ainsi, pour donner un ordre de grandeur, PICASSO dispose d'une soixantaine de fonctions alors que QUANTEL en offre plus de plus de deux cent. Une évolution future de PICASSO se caractériserait donc surtout par un effort dans la création de nouvelles fonctions.

Dans cette optique, on peut faire une remarque importante:
Actuellement, PICASSO vise une clientèle très vaste, ayant des besoins très divers. On pourrait de ce fait diviser les fonctions à ajouter dans deux classes: d'un côté les fonctions de base, nécessaires pour la bonne utilisation du système et d'un autre côté les fonctions particulières, utiles surtout pour un nombre restreint d'applications.

Il n'est pourtant pas toujours aisé de mettre une fonction dans l'une des deux classes, c'est la raison pour laquelle on a adopté une autre classification, en distinguant les fonctions générales de traitement des fonctions concernant les couleurs ou le pinceau.

6.2.1. Fonctions générales

6.2.1.1. Textes

Il existe actuellement une fonction permettant d'insérer du texte dans une image. Or, cette fonction est encore assez rudimentaire, différents besoins n'ayant pas encore été satisfaits.

D'abord, il serait agréable de pouvoir disposer de plusieurs jeux de caractères différents, aux tailles variables. De plus, la possibilité de pouvoir définir soi-même des caractères et de les mémoriser serait un atout important. Cette définition pourrait se faire en utilisant des fonctions de manipulation de caractèress telles que agrandisssement, inclinaison, souligne-ment, aplatissement et autres. Un modèle de la nouvelle police de caractères définie pourra être mis dans un menu contenant l'ensemble des polices existantes.

6.2.1.2. Textures

Le fait de disposer de textures, soit pour le fond de l'image, soit pour remplir des taches particulières, est un avantage important lors de la commercialisation d'une palette. La fonction est très utile en cartographie pour différentier des zones géographiques à l'aide de textures différentes, dans les domaines du textile, du papier peint ou de la publicite.

6.2.1.3. Détourage

Fonction permettant d'isoler un objet par rapport à l'image entière et de ne manipuler que celui-ci. Par extension, plusieurs objets pourraient être groupés ensemble, la manipulation concernant alors tout le groupe. Cette fonction devrait être réalisable, le codage de l'image en mémoire d'entretien à l'aide d'une méthode de Freeman permettant de déterminer les contours et ainsi les objets présents sur l'image.

6.2.1.4. Transformations géométriques

Les plus utilisées sont probablement la translation, la rotation et la mise à l'échelle.

- "La translation définit le déplacement d'un point ou d'un ensemble de points dans le plan ou dans l'espace,
- l'échelle le facteur d'agrandissement d'un point ou d'un ensemble de points, dans le plan ou dans l'espace
- rotation l'action qui consiste à tourner un point ou un ensemble de points, dans le plan ou dans l'espace" [MOUSEL 83].
- Une autre transformation géométrique souvent utilisée est la symétrie. On dit qu'il y a symétrie entre deux points ou deux objets si l'un représente l'image miroir de l'autre, par rapport à un axe ou un plan de symétrie.

Ces fonctions peuvent être globales, si elles s'appliquent à l'image entière, ou locales si la fonction de DETOURAGE permet une application à des objets isolés. Il est évident qu'une telle fonction peut être fort utile en architecture ou en ameublement, lors d'un réarrangement des objets formant l'image.

6.2.1.5. Vue en trois dimensions

Le fait de disposer de la troisième dimension permet de travailler avec des projections perspectives ou parallèles, élargissant ainsi les possibilités de création. Comme, en 3D, les calculs deviennent plus compliqués, un processeur graphique rapide intégrant les transformations géométriques serait souhaitable.

6.2.1.6. Multiplication des images

Fonction générale qui permet de multiplier un objet sur la surface de visualisation. Peut être intéressant dans le cas d'un objet complexe, difficilement reproduisible. Les objets ainsi dédoublés peuvent ensuite être manipulés à l'aide des transformations géométriques.

6.2.1.7. Flou

Dans le cas d'un illustrateur ou d'un publiciste, il se peut que l'on veuille mettre en évidence un objet par rapport au reste de l'image. Cet effet peut être obtenu en entourant cet objet d'un flou, de largeur éventuellement variable, donnant l'impression de brouillard.

6.2.1.8. Animation

Cette fonction, présente dans quelques rares systèmes seulement, n'est généralement requise que pour des applications dans le domaine de l'audiovisuel (surtout la télévision). L'animation en temps réel, c-à-d à la cadence de 25 images/sec, est difficilement réalisable dans le cas d'objets complexes, les temps de traitement devenant trop importants. C'est pourquoi on offre généralement une animation "au ralenti", permettant d'enregistrer les images sur une bande vidéo, la fonction réalisant le déplacement des objets ou de l'image. [JAHIDI 84] donne une introduction à l'animation en général.

6.2.1.9. Anti-aliasing

On définit par aliasing l'effet d'escalier obtenu lors de l'affichage de segments de droites inclinés. Ce défaut résulte en général d'un mauvais échantillonnage des points formant le segment, dû à une résolution trop basse de l'écran ou un calcul trop simplifié des points du segment. Il existe des algorithmes d'anti-aliasing, dont l'utilisation devient nécessaire quand la qualité de l'image doit être très bonne (télévision).

Il serait intéressant de pouvoir inhiber cette fonction lorsqu'un affichage rapide est requis (animation) ou lorsqu'on veut visualiser une ébauche de l'image finale. L'utilité ne se ressent probablement pas si le temps de traitement d'une image n'est pas facturé.

6.2.1.10. Plans d'images

Cette fonction permet de définir plusieurs plans constituant une image ou bien, si on asssimile une image à un plan, de superposer plusieurs images en affichant p.ex. les plans d'ordre inférieur aux plans d'ordre supérieur. Ces plans peuvent être globaux ou locaux de nouveau. S'ils sont globaux, l'effet obtenu est celui de l'existence de plusieurs images en parallèle, pouvant être affichées et modifiées alternativement.

Dans le cas de plans locaux, il faut disposer de la fonction DETOURAGE permettant d'isoler des objets et d'attribuer un niveau à tous les objets. Un changement de plan d'un objet permettrait de le mettre dans l'avant- ou dans l'arrière-plan de l'image.

Une application intéressante de cette fonction serait la réservation des plans de niveau supérieur aux menus, pouvant ainsi être superposés au reste de l'image. Ceci concerne essentiellement le cas d'un système mono-écran, mais peut aussi être intéressant s'il y a deux écrans, dont l'un est dédié aux menus, lesquels peuvent être très nombreux.

6.2.1.11. Aide générale (HELP)

Au fur et à mesure qu'on ajoute de nouvelles fonctions, le système devient plus complexe et l'utilisateur risque de s'y perdre. Dans ce cas, il devient impératif de mettre à sa disposition une fonction d'aide le renseignant à tout instant de la session de travail sur l'état de la machine ou l'utilisation de telle ou telle fonction.

A notre avis, la fonction de détourage et les transformations sont parmi les plus intéressantes, puisqu'elles élargissent considérablement les possibilités de PICASSO, ainsi que la fonction d'aide, améliorant le caractère ergonomique du système.

Ceci dit, nous allons passer maintenant aux fonctions se rapportant aux traitement des couleurs.

6.2.2. Fonctions de manipulation des couleurs

La couleur représente certainement une partie très importante des palettes électroniques. C'est pourquoi on peut envisager l'introduction de diverses fonctions facilitant, ou élargissant, le maniement des couleurs.

6.2.2.1. Agrandissement de la palette visualisée

Bien que cette fonction revête plus de la structure de l'écran, nous avons préféré l'inclure à cet endroit parce qu'elle a une importance capitale dans le maniement des couleurs. En effet, les sept pavés de couleurs ont été jugés insuffisants par la plupart des utilisateurs interrogés [CEFE 84], par rapport aux 256 couleurs différentes disponibles à tout moment, certaines palettes les visualisant toutes. Ce qui paraît être plus intéressant encore à notre avis serait de disposer de pavés vierges, dans lesquels l'utilisateur pourrait "mélanger" ses propres couleurs pour les appliquer au moment voulu. Et pourquoi pas lui donner la possibilité de mémoriser les couleurs ainsi définies, comme on l'a déjà proposé dans les cas des polices de caractères.

6.2.2.2. Dégradé

Fonction fort utile, notamment en cartographie, donnant des couleurs intermédiaires entre plusieurs couleurs différentes, en général deux. Cette fonction peut être très intéressante dans le cas où l'on désire obtenir un effet de "continu", dans les teintes, ce qui crée une image beaucoup plus réaliste que dans le cas d'un coloriage à l'aide d'un nombre restreint de teintes, bien différentiables les unes des autres.

6.2.2.3. Zonification

D'après [CEFE 84], cette fonction est surtout demandée par des décorateurs. Elle consiste à attribuer des codes aux couleurs, de découper l'image en plusieurs parties et d'attribuer alors des codes aux zones ainsi définies. Les zones seront ensuite remplies avec les couleurs correspondantes.

6.2.2.4. Composition d'une couleur

Dans le cas de cette fonction, requise surtout par des cartographes, on permet à l'utilisateur de connaître la quantité des différentes primaires dans une couleur. Celles-ci ne sont pas nécessairement le rouge, vert ou bleu, étant donné qu'en imprimerie, on travaille en quadrichromie. L'application essentielle est de permettre à un imprimeur d'utiliser les mêmes couleurs que celles définies sur l'écran, lors de l'impression d'une carte géographique.

Il ne reste plus qu'à traiter la dernière classe de fonctions que nous présentons dans le cadre des extensions de PICASSO, celles ayant trait au pinceau.

6.2.3. Fonctions du pinceau

6.2.3.1. Aérographe

Cette fonction permet de "vaporiser" en un trait de largeur variable, la couleur courante sur l'écran. La fonction AEROGRAPHE fournit donc un pinceau dont le trait n'est pas uniforme, mais formé d'un grand nombre de petits points, aléatoirement disposés.

6.2.3.2. Pinceau texturé

Dans ce cas, le trait du pinceau est caractérisé par une texture, qui peut être l'une des textures utilisés dans le cadre de la fonction 6.2.1.2.

Conclusion

Nous venons de presenter un certain nombre de caractéristiques interessantes qui pourraient éventuellement faire l'objet d'une extension de PICASSO. Dans le chapitre suivant, on se concentrera plus sur MACCOI, en proposant des extensions possibles relatives aux fonctions qu'on a implémentées.

Chapitre 7 : EVOLUTION POSSIBLE DE MACCOI

Introduction

Dans la partie de PICASSO que nous avons réalisée, on peut distinguer trois niveaux:

- le niveau de la gestion des données
- le niveau du codage
- le niveau de l'optimisation

7.1. Gestion des données

Plusieurs extensions peuvent être intéressantes au niveau de la gestion et de la structuration des données. Voici celles que l'on pourrait imaginer:

7.1.1. Mise-à-jour pendant l'affichage

Nous avons mentionné ce type de mise-à-jour dans le cadre des fonctions de manipulations d'images réalisées. Cette fonction pourrait être utile dans le cas d'une modification du fond de l'image si on ne dispose pas de plans d'image.
Pour pouvoir la réaliser, il faut disposer

- d'un moyen de contrôle de la vitesse d'affichage, pour permettre à l'utilisateur de suivre exactement le processus d'affichage
- d'un moyen d'interruption du défilement, pour indiquer l'endroit de la m-à-j et
- d'un moyen de retour en arrière sans effacement, puisque l'utilisateur ne va probablement se décider de mettre à jour qu'après l'exécution d'une fonction.
- d'un hisorique de la m-a-j. En effet, la m-à-j va modifier l'état du pas d'image interrompu, mais l'état du pas d'image suivant (ou du même, si l'interruption se fait au milieu d'un pas d'image, ce qui compliquera la fonction "arrière) repose sur celui-ci, il est donc nécessaire de pouvoir retourner à l'état au moment de l'interruption, si on veut que l'affichage continue sans erreurs.

On peut raisonnablement se demander si l'utilité de la fonction vaut l'investissement en temps de travail qu'elle nécessite.

7.1.2. Gestionnaire de bibliothèque

Actuellement, il n'y a pas moyen de connaître le contenu d'une bibliothèque à partir de PICASSO. Il est donc impératif de fournir un tel outil à l'utilisateur. On peut distinguer plusieurs fonctions intéressantes dans le cadre d'un tel gestionnaire.

7.1.2.1. Visualisation du contenu

Fonction qui permettrait d'afficher les noms de toutes les images. On pourrait penser à paramétriser la fonction comme c'est le cas dans les systèmes d'exploitation tels que UNIX, pour connaître la date de création, les droits d'accès, etc.

7.1.2.2. Modification des droits d'accès

Cette fonction ne concerne que le cas d'un environnement multi-utilisateurs. Ici peuvent intervenir les notions de propriétaire et de groupe.

7.1.2.3. Création de bibliothèque

Fonction intéressante dans le cas où l'utilisateur désire créer une bibliothèque par type d'images.

7.1.2.4. Description d'image/de bibliothèque

Cette fonction permettrait à l'utilisateur de coller une description, en langage naturel par exemple, à une image ou une bibliothèque. Cette description pourrait être affiché par la fonction de visualisation du contenu.

7.1.2.5. Collage d'images

Cette fonction permettrait de coller plusieurs images ensemble, sans devoir passer par un affichage comme la fonction de superposition.

7.2. Codage

Actuellement, nous ne voyons pas d'extension possible à ce niveau, un taux de compression plus élevé ne pouvant être obtenu que par un investissement intense en temps de travail ne justifiant probablement pas les résultats obtenus.

7.3. Optimisation

L'extension à faire peut se faire sur le premier niveau, optimisation sur les traitements dans les pas d'images, ou sur le deuxième niveau, concernant une optimisation des effets dus aux traitements dans les pas d'images.

7.3.1. Au premier niveau

A ce niveau, l'extension porterait essentiellement sur la palette, les autres aspects du langage ne pouvant plus être traités à ce niveau, leur optimisation étant complète.
Des améliorations peuvent être apportées aux deux aspects de la palette: d'une part l'intensité et la saturation, d'autre part la teinte.

7.3.1.1. Intensité et Saturation

On se souviendra que, lors de la présentation des deux modèles T.I.S. et T.B.N., on a dit que le deuxième résulterait d'un changement d'axes du premier. Analysons les relations entre ces divers aspects de la couleur:
augmenter l'intensité implique un éclaircissement de la couleur, diminuer l'intensité un obscurcissement, donc

$$I = |B - N|$$

De manière analogue, augmenter la saturation entraîne une épuration de la teinte, dès lors

$$S = -|B + N|$$

Supposons que l'utilisateur a surtout travaillé avec B et N, le chemin optimal donnant
nB/signe, mN/signe Soit n > m, on peut réécrire le chemin obtenu de la manière suivante:
(n−m)B/signe, m[B/signe, N/signe]
Quatre cas peuvent se présenter:

m[B/+, N/+] => mS/−
m[B/+, N/−] => mI/+
m[B/−, N/+] => mI/−
m[B/−, N/−] => mS/+

On peut ainsi réduire le nombre d'ordres de moitié, permettant un affichage plus rapide.

7.3.1.2. Teinte

Pour résoudre le problème du chemin minimal, il faut disposer de plusieurs informations :
des valeurs exactes des teintes origine et cible utilisées dans PICASSO. A partir de ces valeurs, on peut déterminer les relations entre les teintes (distance, niveau d'avoisinage).
Les commandes de l'utilisateur ne seront plus utilisées que pour déterminer la teinte cible, le chemin d'accès étant calculé par voie automatique.

Une solution triviale au problème serait de descendre jusqu'au niveau le plus bas et alors de chercher la teinte cible, toutes les teintes étant représentées sur ce niveau. Or, cette solution ne nous donne pas nécessairement un chemin minimal et est donc à rejeter.

L'objectif principal est donc de trouver un chemin d'accès qui minimise le nombre de coordonnées nécessaires. Essayons de trouver une heuristique qui peut nous guider vers la solution. Un principe pourrait être le suivant: comme le nombre de teintes des niveaux supérrieurs est plus petit que celui des niveaux inférieurs, il faut moins de manipulations pour arriver d'une teinte vers une autre; dès lors, le principe serait de se ramener toujours au niveau le plus haut possible. Cette méthode peut fournir un chemin minimal sans le garantir.

Utilisons ce principe pour trouver un chemin d'accès. Pour cela, nous allons distinguer deux cas :

1. la teinte origine est de niveau plus élevé que la teinte cible.
   On essayera d'approcher le plus possible la teinte cible sur le niveau supérieur avant de descendre dans les niveaux Il s'agit donc de trouver la teinte la plus voisine de la cible pour minimiser le rapprochement nécessaire sur le niveau inférieur.
2. la teinte origine est de niveau inférieur à celui de la teinte cible.
   On essayera de monter le plus vite possible au niveau le plus élevé avant d'approcher la teinte par défilement et sélection (‹CHOIX-TEINTE› dans la grammaire du chemin optimal). Dans ce cas, il faut trouver la teinte la plus proche de la teinte origine également présente sur le niveau de la cible.

Ces niveaux ne sont pas nécessairement ceux des teintes au moment de la sélection par l'utilisateur, mais les niveaux les plus hauts possibles pour l'origine et la cible.

Comme cette méthode ne garantit pas un chemin minimal, on peut la raffiner de plusieurs manières:

1. Soit NO le niveau le plus élevé et Nn le plus bas. Si Ti est une teinte particulière du niveau i, elle le sera aussi des niveaux i+1,..,n. On peut dès lors affirmer que la teinte du niveau supérieur se trouve également sur le niveau inférieur, et on peut directement approcher la teinte cible sur le bas niveau et comparer éventuellement le nombre de commandes nécessaires avec celui qu'il faut avec la méthode proposée.
2. On peut aussi calculer un chemin minimal par niveau

intermédiaire entre le plus haut et le plus bas et même comparer avec l'approche directe, sur un niveau.
Plus on raffine, plus les calculs deviennent longs, mais plus on a de chances de trouver le chemin minimal.

7.3.2. Au deuxième niveau

Une autre extension de la fonction d'optimisation pourrait porter sur le deuxième niveau qu'on a défini: celui d'une optimisation des effets. Ce niveau d'optimisation nécessite une approche plus délicate que le premier, puisqu'on intervient directement sur l'image à afficher, en la modifiant. Plutôt que d'essayer de trouver une solution à ce problème, solution qui n'existe peut-être même pas, nous allons mettre en évidence plusieurs questions auxquelles il faudra répondre avant de pouvoir optimiser quoi que ce soit.

Que faut-il optimiser?

La question pourrait aussi être: quelles sont les actions qui doivent être optimisées parce que jugées superflues ou contradictoires. Donnons un exemple:
Dans le cas où l'image finale est la seule importante, la seule couleur de fond intéressante est la dernière, toutes les modifications précédentes devenant superflues. Il en est de même lors du recouvrement complet d'un objet par un autre, le premier ne devant plus être dessiné.
Le "que optimiser" dépend donc fortement de ce qui est l'objectif du peintre, le changement du fond pouvant être un effet voulu dans une application d'animation. Une deuxième question se pose alors:

Quand faut-il optimiser?

Une réponse à cette question pourrait être: lorsque l'exécution de l'image non-optimisée est moins performante en temps de calcul et en occupation mémoire que l'image optimisée. Cette façon de raisonner implique qu'il faut disposer d'un moyen de déterminer la complexité de l'image. Il faudrait donc procéder à une optimisation dans le cas d'un effacement quasi-global d'un objet desssiné en trait continu et ne pas optimiser lorsque seulement une petite partie de l'objet est effacé, l'effacement prenant moins de temps que le dessin de l'objet modifié.
Si un tel problème peut encore être résolu par calcul, il devient indécidable dans le cas où la volonté de l'utilisateur intervient. Dans ce cas, l'optimisation devrait donc se faire en interactif, le système proposant à l'utilisateur des optimisations éventuelles et lui laissant le choix de les faire ou non.

Finalement, on peut dire que cette optimisation deviendra certainement seulement possible si la notion d'objet isolé existe, permettant d'enlever ou de modifier un objet sur l'écran.

CONCLUSION GENERALE

Dans les trois parties du présent exposé, nous nous sommes efforcés de décrire d'un côté la partie existante du système PICASSO en le situant par rapport aux différents champs de l'informatique dans lesquels le traitement de données graphiques est prioritaire, d'un autre côté de présenter la partie d'ajout que nous avons réalisée dans le cadre de ce mémoire et de proposer finalement quelques extensions possibles.

Le lecteur intéressé trouvera en annexe une partie importante de l'analyse effectuée pour mener à bien la réalisation des différentes fonctions.

BIBLIOGRAPHIE

[ANDREWS 76] : H.C. Andrews & C.L. Patterson
"Outer Product Expansions and their Uses in Digital Image Processing"
IEEE Transactions on Computers, Vol C-25, No 2, February 1976

[BODART 83] : François Bodart
"Analyse fonctionnelle"
Cours de le licence en informatique,
Facultès Universitaires de Namur, 1983

[BRADY 82] : Michael Brady
"Computational Approaches to Image Understanding"
Computing Surveys, Vol 15, No 1, March 1982

[CAPON 59] : Jack Capon
"A Probabilistic Model for Run-Length Coding of Pictures"
IRE Transactions on Information Theory, pp.157 - 163, December 1959

[CEPE 84] : Centrale d'Evaluation et de Faisabilite Economique
"Etude de Valorisation du Produit PICASSO"
Centre de Recherches d'Economie d'Entreprises,
Lille Septembre 1984

[CHARROUF 84] : Khalid Charrouf
"Etude et Réalisation du Système Graphique Interactif PICASSO"
Thèse de docteur de troisième cycle,
Université des Sciences et Techniques de Lille, 1984

[DYER 80] : Charles E. Dyer
"Space Efficiency of Region Representaton by Quadtrees"
IEEE, 1980

[FIRSCHEIN 72] : O. Firschein & M.A. Fischler
"A Study in Descriptive Representation of Pictorial Data"
Pattern Recognition, Vol. 4, pp. 361 - 377, 1972

[FOLEY 82] : James D. Foley & Andries Van Dam
"Fundamentals of Interactive Computer Graphics"
Addison Wesley, 1982

[GIAMBIASI ..] : Norbert Giambiasi, Jean-Claude Rault &
Jean-Claude Sabonnadière
"Introduction à la Conception Assistée
par Ordinateur"
..

[JAHIDI 84] : Khalil Jahidi
"Modèle de Contrôle et d'Animation de
Systèmes Mécaniques pour l'Informatique
Graphique"
Thèse de docteur de troisième cycle,
Université des Sciences et Techniques de Lille, 1984

[KOBAYASHI 74] : H. Kobayashi & L.R. Bahl
"Image Data Compression by Predictive Coding"
IBM Journal of Research and Development,
March 1974

[LUCAS 82] : Michel Lucas
"La Réalisation des Logiciels Graphiques Interactifs"
Ed. Eyrolles 1982

[MERIAUX 79] : Michel Meriaux
"Etude et Réalisation d'un Terminal Graphique
Couleur Tridimensionnel Fonctionnant par Tâches"
Thèse de docteur ingénieur; Université des
Sciences et Techniques de Lille, 1979

[MEYER 83] : Dietrich Meyer-Ebrecht & Thomas Wendler
"An Architectural Path through PACS"
Computer, August 1983

[MORVAN 76] : Pierre Morvan & Michel Lucas
"L'Infographie Interactive"
Larousse, 1976

[MOUSEL 83] : Aurora Mousel & Michèle Baum
"Mathématiques et Programmation des
Projections Géométriques Planaires"
Mémoire de Graduat en Dessin d'Architecture,
Institut St. Luc Bruxelles 1983

[NEWMAN 81] : William M. Newman & Robert F. Sproull
"Principles of Interactive Computer Graphics"
Mc Graw Hill, 1981

[OLIVER 83] : M.A. Oliver & N.E. Wiseman
"Operations on Quadtree Encoded Images"
The Computer Journal, Vol 26, No 1, 1983

[ROBERTS 62] : Lawrence G. Roberts
"Picture Coding Using Pseudo-Random Noise"
IRE Transactions on Information Theory, pp.
145 - 154, 1962

[ROESE 77] : J.A. Roese, W.K. Pratt & G.S. Robinson
"Cosine Transform Image Coding"
IEEE Transactions on Computers, Vol C-25, No 11,
November 1977

[SAMI 73] : M. Sami & R. Stefanelli
"Compression Algorithms that Preserve Basic
Topological Features in Binary-Coded Patterns"
Pattern Recognition, Vol 5, pp. 133 - 147, 1973

[STEVENS 81] : Stevens
"Using structured design"
A Whiley Interscience Publication
John Whiley and Sons, 1981

[SUTHERLAND 63] : Ivan E. Sutherland
"SKETCHPAD : A Man Machine Graphical Communication
System"
Conference Proceedings, Spring Joint Computer
Conference 1963

[VAN LAMSWEERDE 84] : Axel Van Lamsweerde
"Génie logiciel"
Cours de 2e licence en informatique,
Facultès Universitaires de Namur, 1984

[WELCH 84] : Terry A. Welch
"A Technique for High-Performance Data Compression"
Computer, June 1984

[WHOLEY 61] : Joseph S. Wholey
"The Coding of Pictorial Data"
IRE Transactions on Information Theory, pp.
99 - 104, 1961

[WOODWARK 84] : J.R. Woodwark
"Compressed Quadtrees"
The Computer Journal, Vol 27, No 3, 1984

[ZAHN 72] : C.T. Zahn & R.Z. Roskoes
"Fourier Descriptors for Plane Closed Curves"
IEEE Transactions on Computers, Vol C-21, No 3,
March 1972

ANNEXES

ANNEXE 1 : ANALYSE DU PROBLEME

Introduction

Dans cette première annexe, je vais essayer de spécifier le problème et d'en donner le design, du moins partiellement, de façon formelle (par opposition aux spécifications et solutions informelles données dans la partie II du mémoire).

1.1. Analyse des besoins

Je ne vais pas y revenir, les besoins ayant été explicitement exprimés dans les deux premières parties.

1.2. Spécifications

1.2.1. Identification des phases du traitement

Le logiciel qui faisait l'objet de mon travail traite des images. J'ai donc identifié plusieurs phases de traitement, correspondant au cycle de vie d'une image. (*)

Cycle de vie d'une image

1. Création d'une image
2. Mémorisation d'une image
3. Codage d'une image
4. Optimisation d'une image
5. Décodage d'une image
6. Mise-à-jour d'une image

Remarque : Le lecteur averti pourrait se poser la question pourquoi on a optimisé après le codage, entraînant ainsi une complication de l'optimiseur comme il devra manipuler deux types d'informations différentes (codes et coordonnées) au lieu d'un seul. La raison en est une plus grande indépendance vis-à-vis de la structure de l'écran

(*) Le concept de phase ici utilisé est légèrement plus vaste que celui utilisé dans [BODART 83], l'unité spatio-temporelle n'étant pas toujours présente. J'ai choisi cette découpe puisquelle me permet de regrouper des traitements analogues, composés éventuellement de plusieurs phases au sens de [BODART 82].

graphique, l'avantage du codage étant qu'il ne porte aucune information sur l'emplacement de l'ecran qu'il réfère.

PHASE 1 : CREATION d'une image
Cette phase correspond en fait à PICASSO, auquel est ajouté un fichier graphique contenant une description on-line de l'image (par opposition à l'image stockée) et une fonction d'effacement automatique.

Fonction 1.1 : Arrière (Effacement automatique)
Fonction 1.2 - 1.n : foncions de PICASSO

PHASE 2 : MEMORISATION d'une image
Cette phase correspond au stockage effectif du fichier graphique on-line. Elle n'a pas de sous-fonctions.

PHASE 3 : CODAGE d'une image
A cette phase correspond le passage de l'image stockée sous forme non-codée vers une représentation codée.

Fonction 3.1 : Codage du pinceau
Fonction 3.2 : Codage de la palette
Fonction 3.3 : Codage du menu des fonctions

PHASE 4 : OPTIMISATION d'une image
Dans cette phase du traitement, on passe d'une représentation codée non optimisée vers une représentation codée: optimisée.

Fonction 4.1 : Validation du symbole (code/coordonnée)
Fonction 4.2 : Optimisation du pinceau
Fonction 4.3 : Optimisation de la palette
Fonction 4.4 : Optimisation du menu des fonctions
Fonction 4.5 : Optimisation des coordonnées

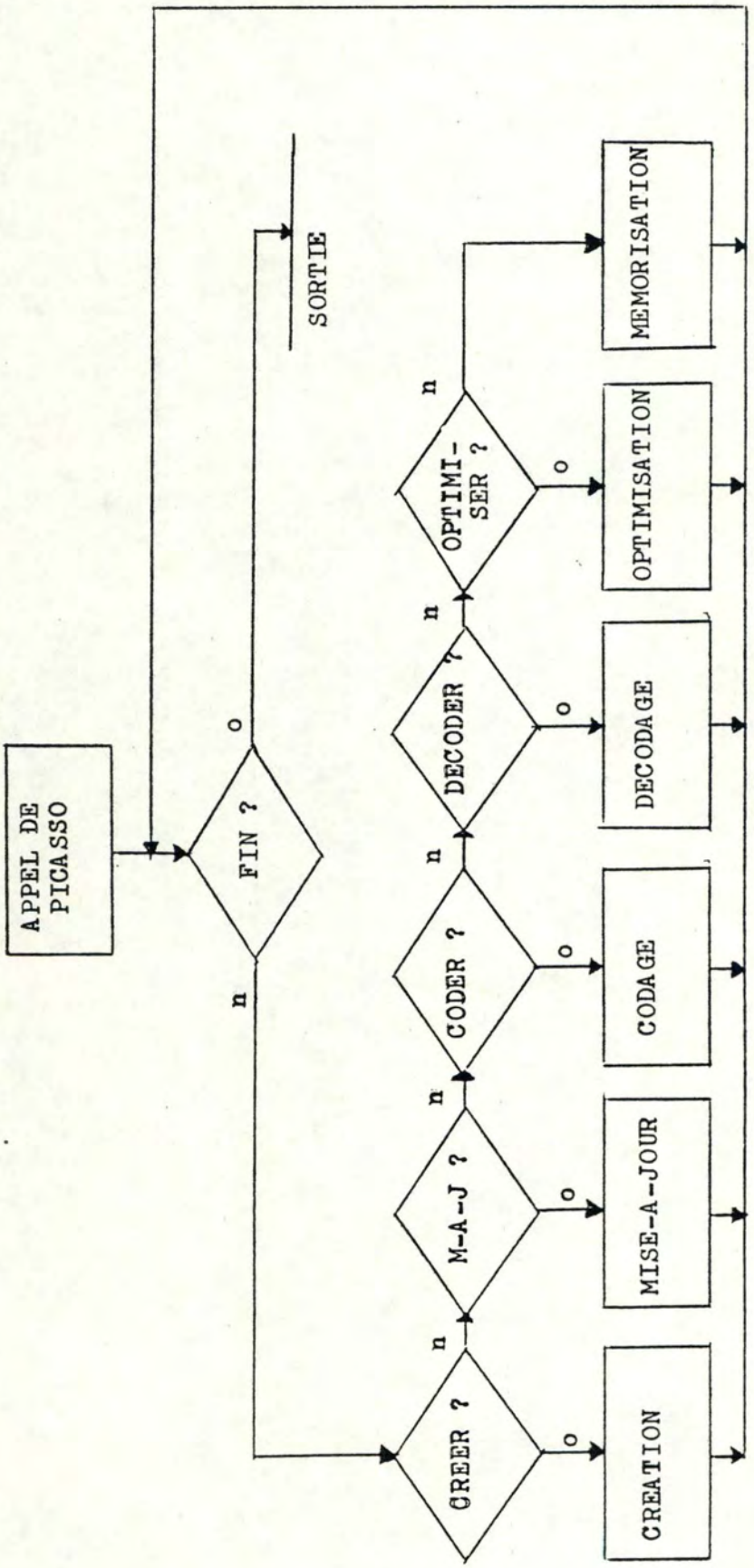
PHASE 5 : DECODAGE d'une image
On passe d'une représentation codée vers une représentation non codée, exécutable.

Fonction 5.1 : Décodage du pinceau
Fonction 5.2 : Décodage de la palette
Fonction 5.3 : Décodage du menu des fonctions

PHASE 6 : MISE-A-JOUR d'une image
Cette phase correspond à une modification interactive de l'image.

Fonction 6.1 : Destruction de l'image
Fonction 6.2 : Renommer une image
Fonction 6.3 : Affichage d'une image
Fonction 6.4 : Superposition de deux images

1.2.2. Dynamique des phases



1.2.3. Structuration hiérarchique de MACCOI

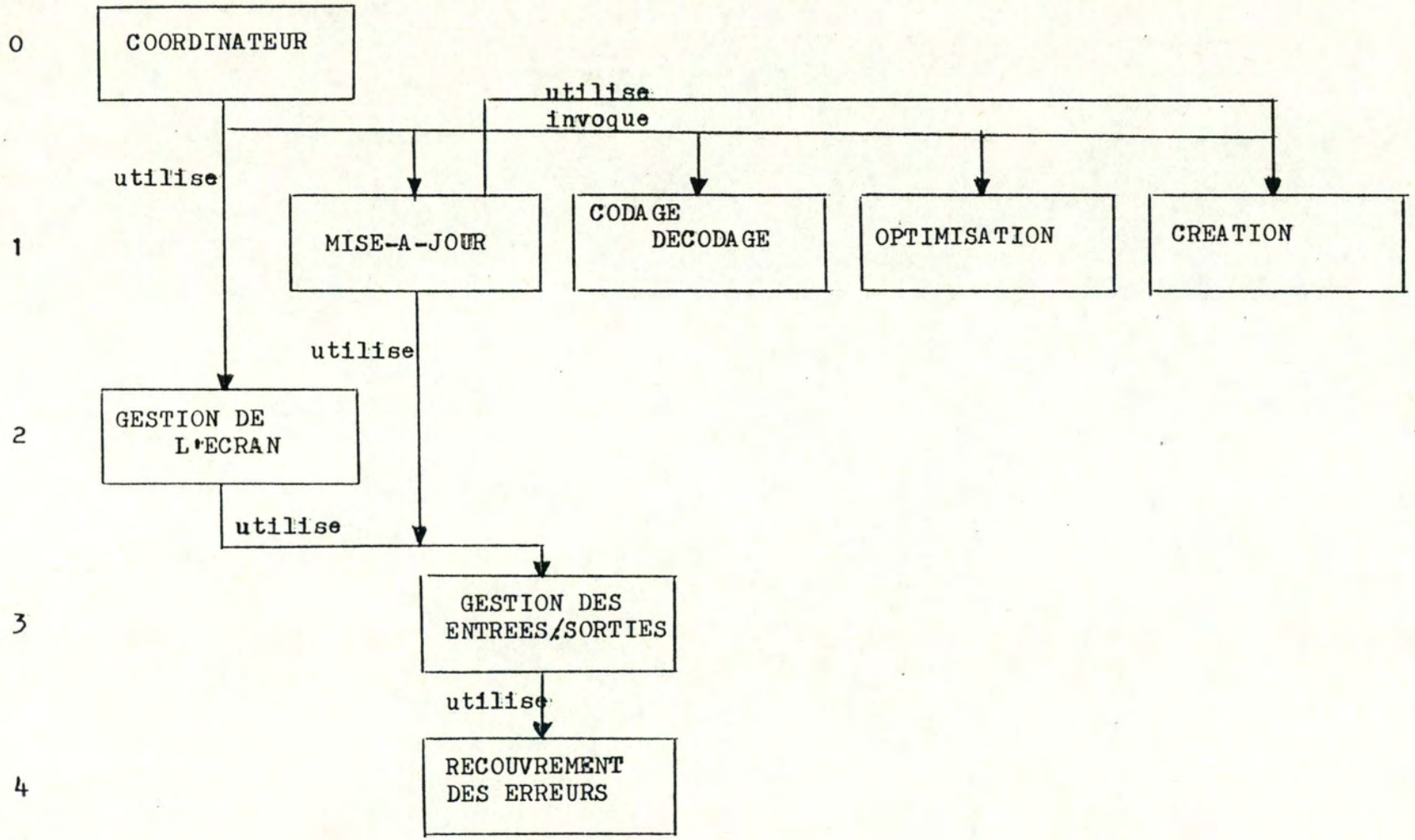
1.2.3.1. Identification des composants

Le logiciel est formé de plusieurs composants, correspondant aux phases définies au point précédent, où j'ai regroupé codage et décodage, et ajouté un coordinateur.

1.2.3.2. Relations entre composants

Selon [VAN LAMSWEERDE 83], les composants d'un logiciel ont généralement des relations entre eux, telle que les relations utilise, importe, exporte, invoque ou autres.
[VAN LAMSWEERDE 83] définit la relation utilise de la façon suivante:
Soient A et B deux composants du système,on dit que A utilise B si le fonctionnement correct de A dépend du fonctionnement correct de B, si la réalisation de A devient plus simple du fait d'utiliser B et que la réalisation de B ne devient pas plus compliqué du fait de ne pas utiliser A.
Ce sera là une relation importante entre les composants du logiciel.
Il y a en plus la relation invoque, dont la spécification est identique à celle de utilise moyennant le fait que dans A invoque B, le fonctionnement correct de A ne dépend pas de B.

NIVEAU :


0
COORDINATEUR
utilise
invoque
utilise
1
MISE-A-JOUR
CODAGE
DECODAGE
OPTIMISATION
CREATION
utilise
2
GESTION DE
L'ECRAN
utilise
3
GESTION DES
ENTREES/SORTIES
utilise
4
RECOUVREMENT
DES ERREURS

1.2.4. Modularisation du logiciel

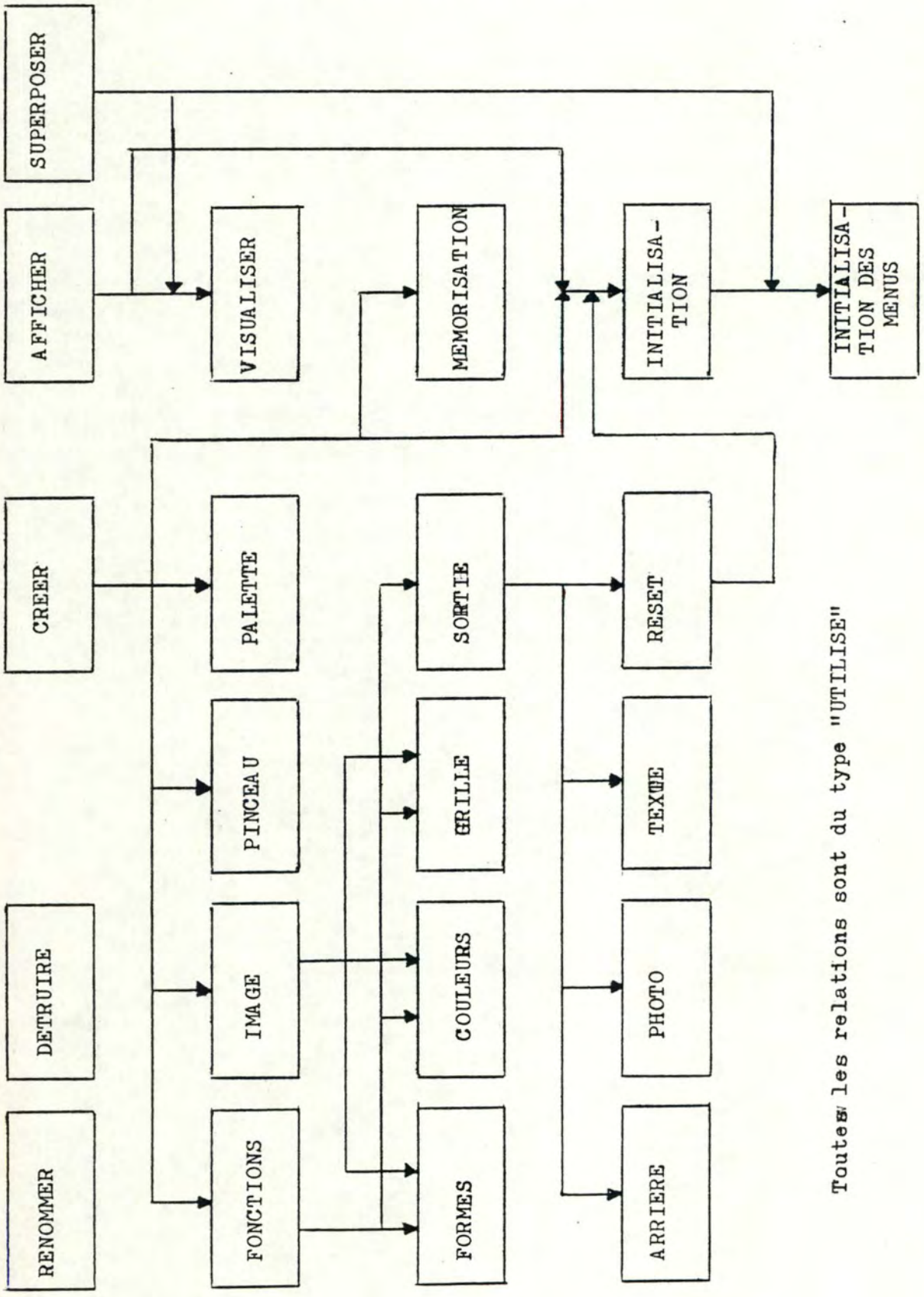
1.2.4.1. Définition de module

Un module est un composant d'un système dont les attributs sont définissables de manière simple et qui offre les qualités suivantes:

1. forte capacité de cacher de l'information (principe de la boîte noire)
2. forte cohésion interne (binding).
   Ceci indique que tous les éléments d'un module sont fortement liés, la liaison fonctionnelle étant la plus forte (les éléments concernent tous une même fonction) [STEVENS 81].
3. faible degré de couplage (coupling).
   Cette qualité indique que les relations entre les modules sont très faibles.

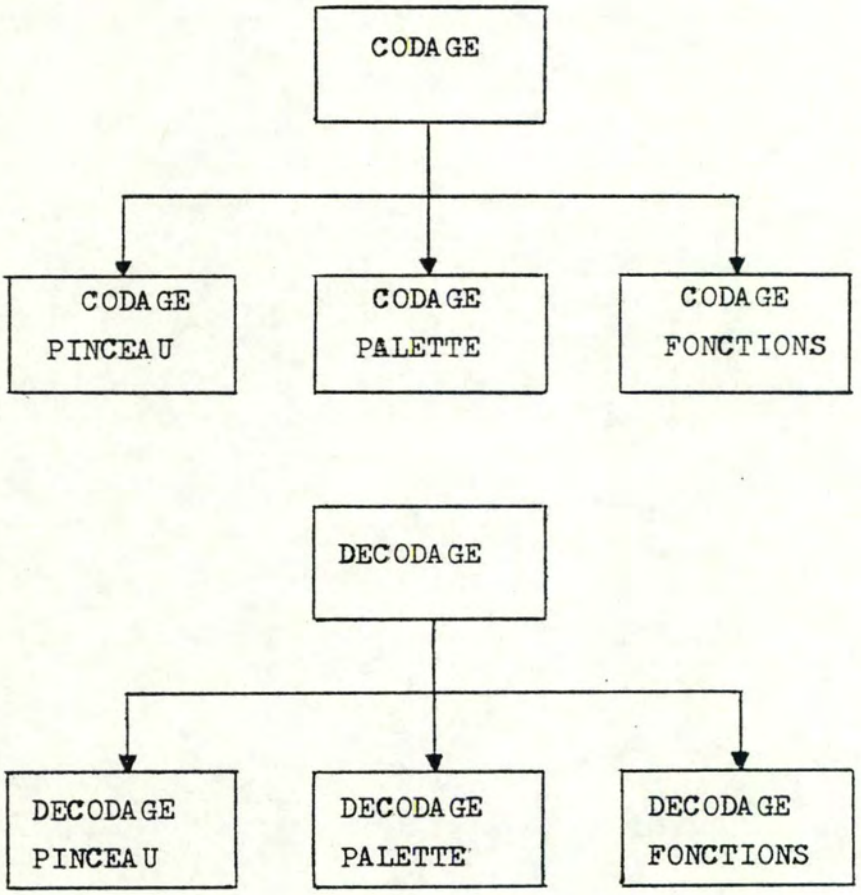
1.2.4.2. Structuration du logiciel en modules fonctionnels

- Création/Stockage/Mise-à-jour



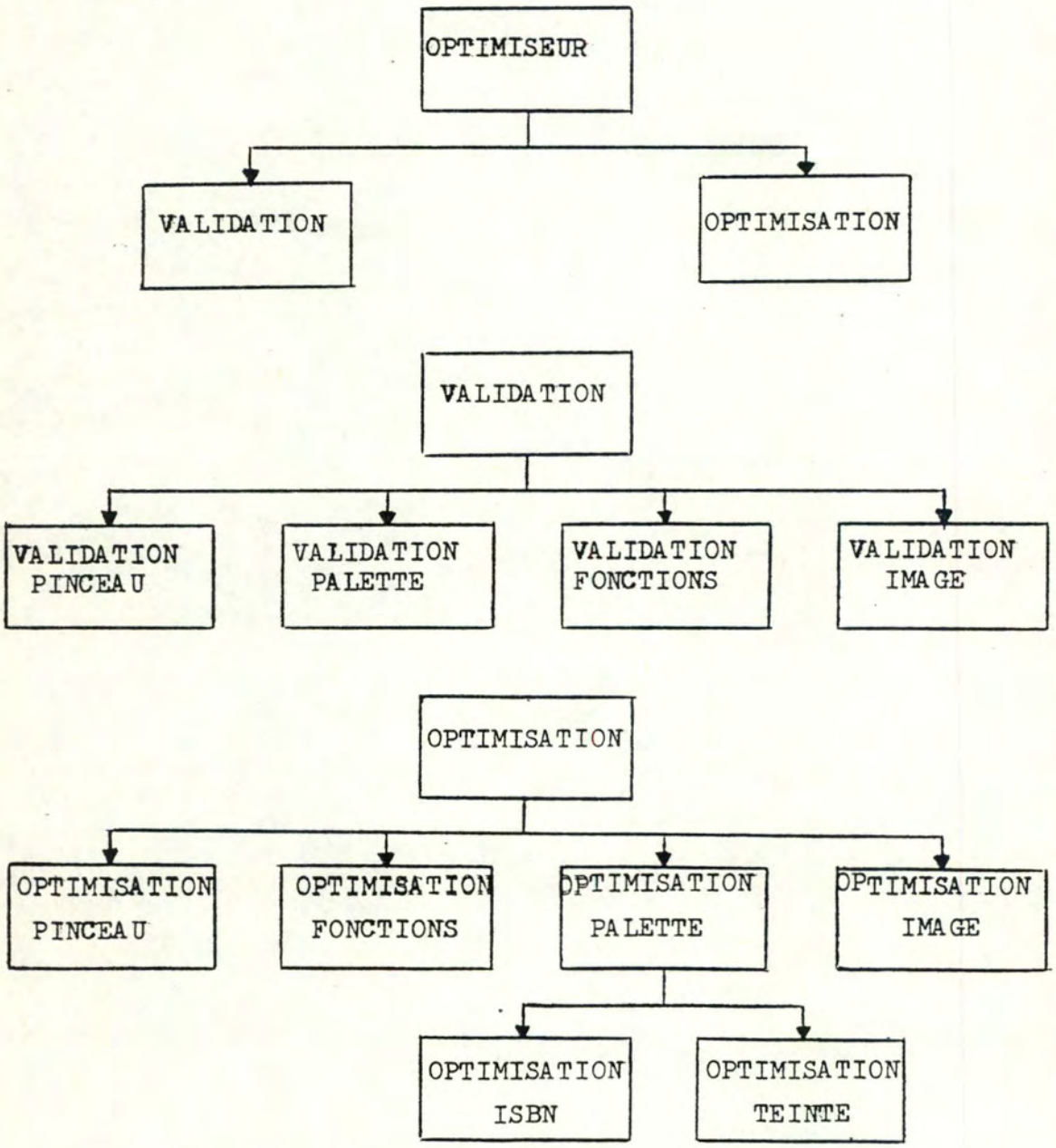
Toutes les relations sont du type "UTILISE"

- Codage/Decodage



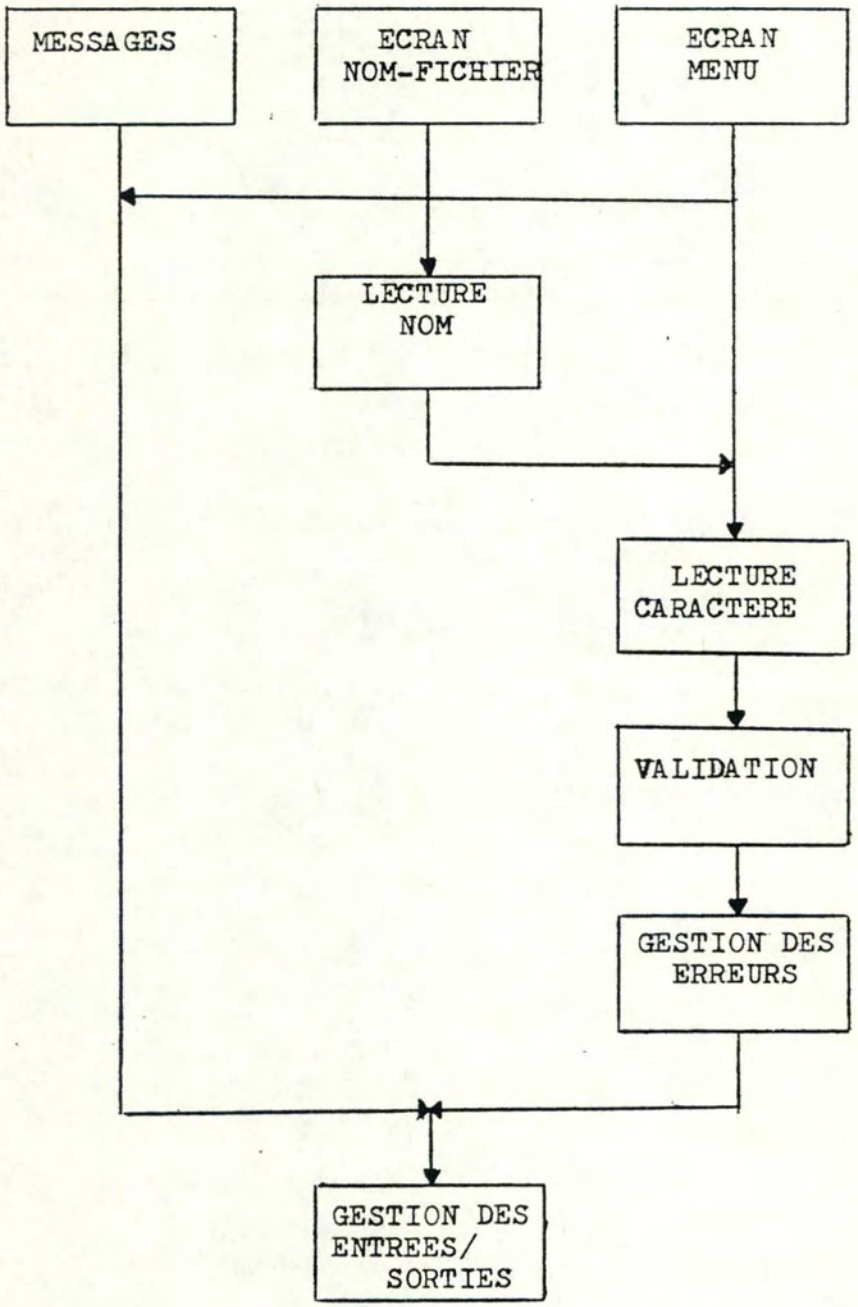
Toutes les relations sont du type "UTILISE"

- Optimisation



Toutes les relations sont du type "UTILISE"

- Gestion de l'écran



Toutes les relations sont du type "UTILISE"

1.2.4.3. Spécifications abstraites

[VAN LAMSWEERDE 83] définit par spécifications abstraites d'un module les informations nécessaires sur celui-ci pour pouvoir l'utiliser convenablement. On distingue les spécifications abstraites des spécifications concrètes, ces dernières donnant en plus des informations sur les types de données utilisés. Comme je ne manipule pas beaucoup de données différentes, les variables mises à part, j'indiquerai les structures des données lors du design.

Les spécifications abstraites seront données par précondition, indiquant les conditions que les arguments doivent vérifier, et postcondition, indiquant les conditions à vérifier par les résultats.

1. Module COORDINATEUR

```
Arguments      : Identifiant d'un composant du niveau 1
Précondition   : Identifiant valide
  Résultat     : déclenchement de l'exécution du composant
                 sélectionné
Postcondition  : /
```

2. Module MISE-A-JOUR

2.1. Fonction AFFICHAGE

```
Arguments      : nom-image
Précondition   : (nom-image valide) & (∃ ! Image(:Nom = nom-image)) &
                    (Ecran-graphique vide) & (Etat-pinceau = mince) &
                    (Etat-menu = continu) & (Etat-grille = absent) &
                    (Etat-palette = (bleu,min-saturation))
  Résultat     : Ecran-graphique =
                      Ecran-graphique + Image(:Nom = nom-image)
Postcondition  : ∃ ! Image(:Nom = nom-image)
```

2.2. Fonction SUPERPOSITION

```
Arguments      : nom-image
Précondition   : (nom-image valide) & (∃ ! Image(:Nom = nom-image)) &
                    (Etat-pinceau = mince) & (Etat-menu = continu) &
                    (Etat-grille = absent) &
                    (Etat-palette = (bleu,min-saturation))
  Résultat     : Ecran-graphique =
                      Ecran-graphique + Image(:Nom = nom-image)
Postcondition  : ∃ ! Image(:Nom = nom-image)
```

2.3. Fonction DESTRUCTION

```
Arguments      : nom-image
Précondition   : (nom-image valide) & ( ∃ ! Image(:Nom = nom-image)
  Résultat     : Bibliothèque-images =
                     Bibliothèque-images - Image(:Nom = nom-image)
Postcondition  : ∄ Image(:Nom = nom-image)
```

2.4. Fonction RENOMMER

```
Arguments      : old-nom, new-nom
Précondition   : ((old-nom & new-nom) valide) &
                    (∃ ! Image(:Nom = old-nom)) &
                    (∄ Image(:Nom = new-nom))
  Résultat     : Image(:Nom = new-nom)
Postcondition  : (∃ ! Image(:Nom = new-nom)) &
                 (∄ Image(:Nom = old-nom))
```

3. Module CODAGE/DECODAGE

3.1. Fonction CODAGE

```
Arguments      : nom-image
Précondition   : (nom-image valide) &
                    (∃ ! Image(:Nom = nom-image) &
                    (Image(:Nom = nom-image) non-codée)
  Résultat     : Image(:Nom = préfixe(nom-image).COD)
Postcondition  : Image(:Nom = préfixe(nom-image).COD) codée
```

3.2. Fonction DECODAGE

```
Arguments      : nom-image
Précondition   : (nom-image valide) &
                    (∃ ! Image(:Nom = nom-image)
  Résultat     : Image(:Nom = préfixe(nom-image).DEC)
Postcondition  : Image(:Nom = préfixe(nom-image).DEC) décodée
```

4. Module CREATION

4.1. Fonction ARRIERE

```
Arguments    : I = (PI1,PI2,..,PIi,..,PIn)
Précondition : 0 < i <= n,   k t.q. i+1 <= k <= n, PIk incomplet
     <=> PIk = CXT [vide | {P}0->1 | {P}0->2]
        i t.q. PIi complet
     <=> PIi = CXT [{P}1->m |
                    {PP}1->m {P}0->1 |
                    {PPP}1->m {P}0->2]      m > 1
  Résultat     : I = (PI1,PI2,..,PI'i,..PIn)
     PI'i = CXT [{P}1->m-1 |
                 {PP}1->m-1 |
                 {PPP}1->m-1]               m >= 1
Postcondition : /
```

5. Module OPTIMISEUR

5.1. Fonction VALIDATION

```
Arguments    : code | coordonnée
Précondition : /
  Résultat     : valide(code | coordonnée) |
                 invalide(code | coordonnée)
Postcondition : /
```

5.2. Fonction OPTIMISATION

```
Arguments    : PI
Précondition : PI complet
  Résultat     : PI' = optim(PI)
Postcondition : (effet(PI') = effet(PI)) & (PI' ⊂ PI)
```

5.2.1. Fonction O-PINCEAU

```
Arguments    : contexte-pinceau
Précondition : contexte-pinceau <> vide
  Résultat     : contexte-pinceau' =
                     chemin-opt(contexte-pinceau)
Postcondition : (effet(contexte-pinceau') =
                 effet(contexte-pinceau)) &
                (contexte-pinceau' ⊂ contexte-pinceau)
```

5.2.2. Fonction O-PALETTE

```
Arguments      : contexte-palette
Précondition   : contexte-palette <> vide
  Résultat     : contexte-palette' =
                      chemin-opt(contexte-palette)
Postcondition  : (effet(contexte-palette') =
                  effet(contexte-palette)) &
                 (contexte-palette' C contexte-palette)
```

5.2.3. Fonction O-MENU

```
Arguments      : contexte-menu
Précondition   : contexte-menu <> vide
  Résultat     : contexte-menu'
Postcondition  : (effet(contexte-menu') =
                  effet(contexte-menu)) &
                 (contexte-menu' C  contexte-menu)
```

5.2.4. Fonction O-IMAGE

```
Arguments      : INPUT = [{P}1->n |
                          <PP> {P}0->1 |
                          <PPP> {P}0->2]
Précondition   : "implicitement donnée par VALIDE"
  Résultat     : OUTPUT = [{P}1->n |
       état-grille(adapt-grille(<PP>),<PP>) |
       état-grille(adapt-grille(<PPP>),<PPP>)]
Postcondition  : effet(INPUT) = effet(OUTPUT)
```

1.3. Design (Extraits)

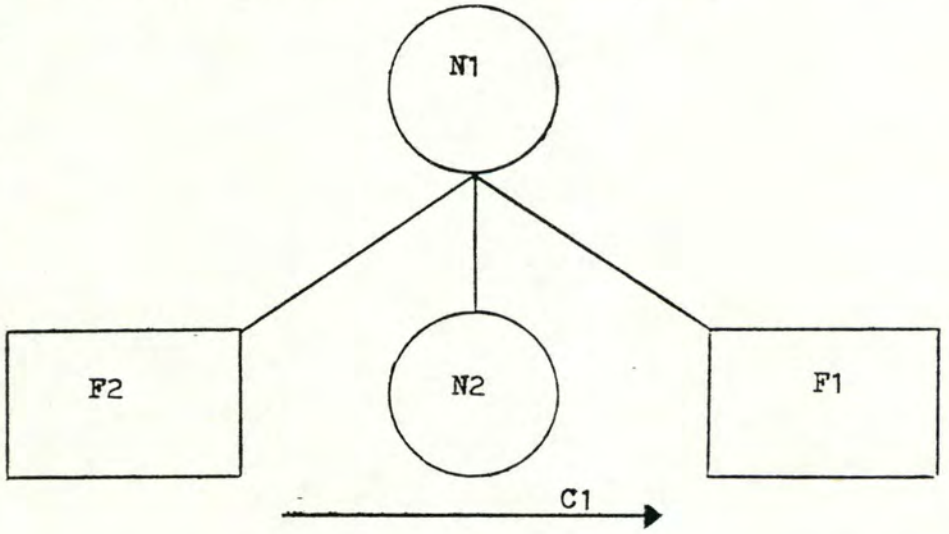
Je ne reprendrai dans cette section que les modules les plus interessants ou difficiles, permettant d'obtenir une vision plus detaillee de l'implementation. En ce qui concerne les modules ou fonctions non detailles, je juge les extraits donnees ici, en commun avec le code documente, suffisamment parlant pour ne pas devoir tout detailler ici.

La partie du design est divisee en trois sous-parties, la premiere ayant trait aux images interactives, donc on-line, la deuxieme aux images stockees, donc off-line, et la derniere aux outils utilises par les deux autres.

1.3.1. Representation des donnees

Les structures des donnees seront indiquees a l'aide de la methode de BERTHINI et TALLINEAU, dont je rappelle brievement les concepts ici:

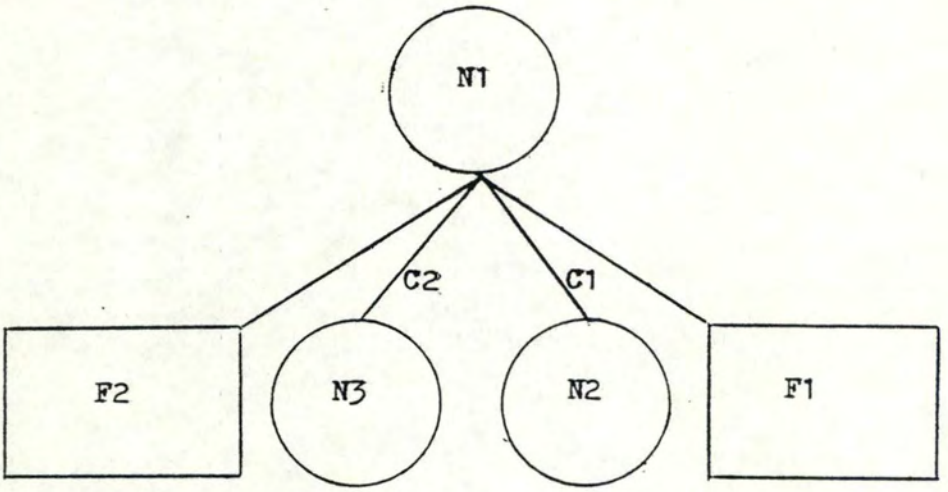
1. La structure des donnees sera representee sous forme d'arbre, formé de deux symboles: un rectangle indiquant une feuille; un cercle indiquant un noeud (terminal ou non).
2. L'arbre ne peut contenir que deux composants differents: soit une repetitive, representee comme suit:



dont l'equivalent en pseudo-langage serait:

```
N1:
  faire F1;
  tant que pas C1;
      faire N2;
  fin;
  faire F2;
fin;
```

soit une alternative, dont la representation peut être comme suit:



dont l'equivalent en pseudo-langage serait:

C2 = non C1

```
N1:
  faire F1;
  si C1
    alors faire N2;
    sinon faire N3;
  faire F2;
fin;
```

C2 <> non C1

```
N1:
  faire F1;
  si C1
    alors faire N2;
    sinon si C2
            alors faire N3;
  faire F2;
fin;
```

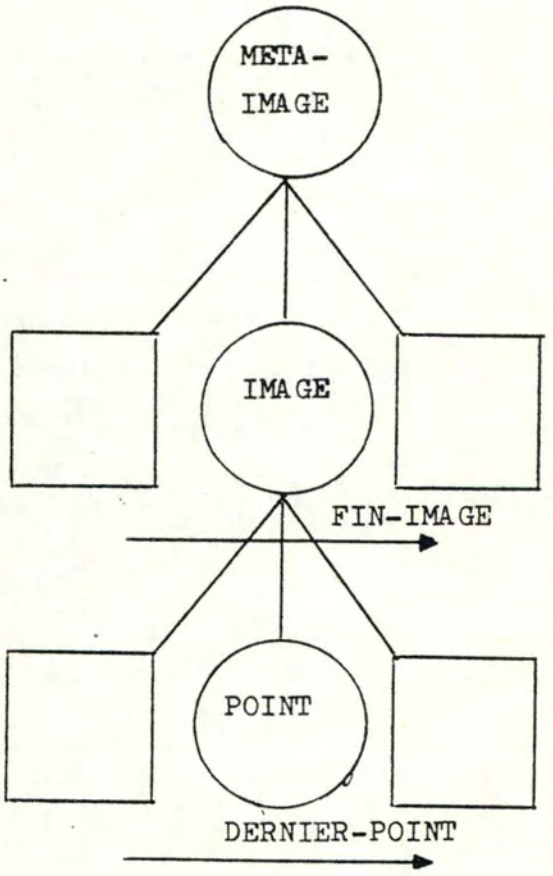
1.3.2. Module fonctionnel MIDESS

Le module est fonctionnel parce qu'il regroupe plusieurs fonctions apparentées, ce sont la création, le stockage et la mise-à-jour qui se font tous sur une image on-line.

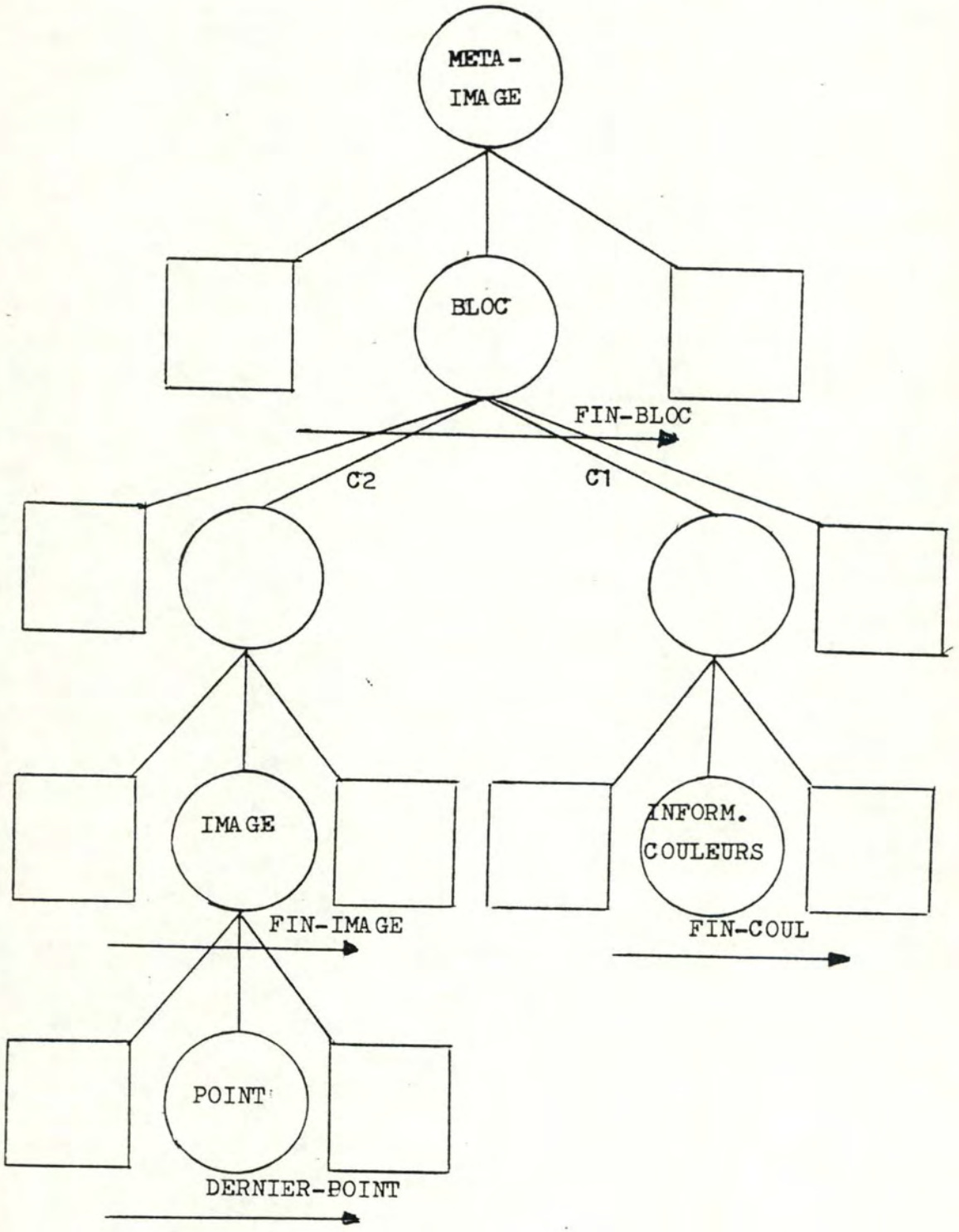
1.3.2.1. Structure des données

- Logique

  La structure logique la plus "évidente" d'une image est probablement la suivante:



- Logico-physique
  La structure physique étant importante lors de l'implémentation, je l'ai mise en commun avec la structre logique pour donner une structure globale, par module fonctionnel puisque la structure logique peut varier.



C1 : Informations-suivantes $\stackrel{?}{=}$ Description-couleur

C2 : Informations-suivantes = Description-image

Les informations sur les couleurs sont d'une part nécessaires dans l'optimisation, pour garder une certaine indépendance par rapport au traitement des couleurs de PICASSO, et d'autre part pour mettre le maximum d'informations sur une image dans le fichier, permettant ainsi d'autres applications, comme une étude sur les couleurs.
Les informations minimales sur les couleurs nécessaires sont celles concernant la fonction COULEUR-SUR-IMAGE, puisque l'image stockée ne contient aucun renseignement sur la disposition des couleurs sur l'écran.

1.3.2.2. Le module principal

```
MIDESS:
     Initialisation-globale ;
     tant que pas Fin-trt faire ;
          demander Choix-trt ;
          selon Choix-trt faire ;
               CREER ;
               AFFICHER ;
               SUPERPOSER ;
               DETRUIRE ;
               RENOMMER ;
               STOCKER ;
          fin ;
     fin ;
     Terminaison-globale ;
fin;
```

1.3.2.3. Effacement automatique

Le lecteur se référera à la structure des données de l'optimiseur pour bien comprendre cette fonction.

```
ARRIERE de CREER ;
     Initialisation ;
     tant que pas (Trouve-PI-complet ou Début-image) faire
          Trouvé-fonction = RECH-FONCTION(Borne-sup,Borne-inf) ;
          si pas Trouvé-fonction
             alors Début-image ;
                   Fonction = continu ;
          Nbre-coord-min = F(Fonction) ;
          PI-complet = RECH-PARAMETRES ;
      fin ;
      si PI-complet
         alors ENLEVER-PARAMETRES ;
fin ;
```

```
RECH-FONCTION(Borne-sup,Borne-inf) ;
  (* Borne-sup et Borne-inf déterminent les bornes
     de l'intervalle dans lequel le pas d'image se trouve *)
     tant que pas (Action(Borne-inf) ou (Borne-inf = min)) ;
          faire RECULER(Borne-inf) ;
          si Menu(Borne-inf)
             alors Borne-sup = Borne-inf ; (* PI incomplet *)
     fin ;
     si Action(Borne-inf)
        alors vrai ;
        sinon faux ;
fin ;
```

1.3.2.4. Afficher une image

- Afficher

```
AFFICHER ;
     Init-globale-PICASSO ;
     VISUALISATION ;
fin ;
```

- Superposer

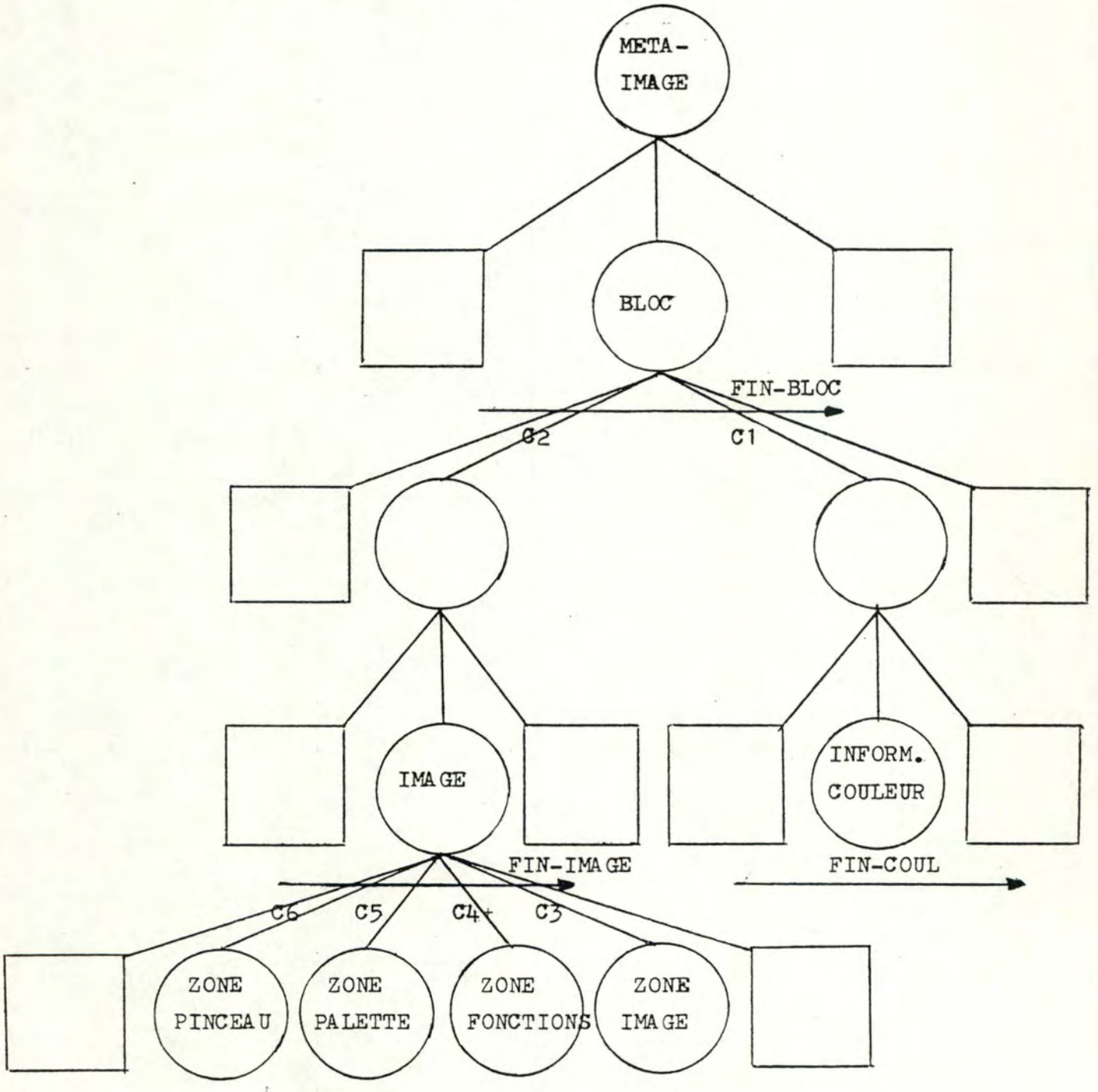
```
SUPERPOSER ;
     Init-menus-PICASSO ;
     VISUALISATION ;
fin ;
```

- Visualisation

```
VISUALISATION :
     lire Image-stockée ;
     tant que pas Fin-image faire ;
          DESSINER(coordonnée) ;
          lire Image-stockée ;
     fin ;
fin ;
```

1.3.3. Module fonctionnel CODECO

1.3.3.1. Structure des donnees



C1 : Informations-suivantes $\stackrel{?}{=}$ Description-couleurs

C2 : Informations-suivantes $\stackrel{?}{=}$ Description-image

C3 : Point de la zone image ?

C4 : Point de la zone du menu des fonctions ?

C5 : Point de la zone de la palette ?

C6 : Point de la zone du pinceau ?

1.3.3.2. Codage

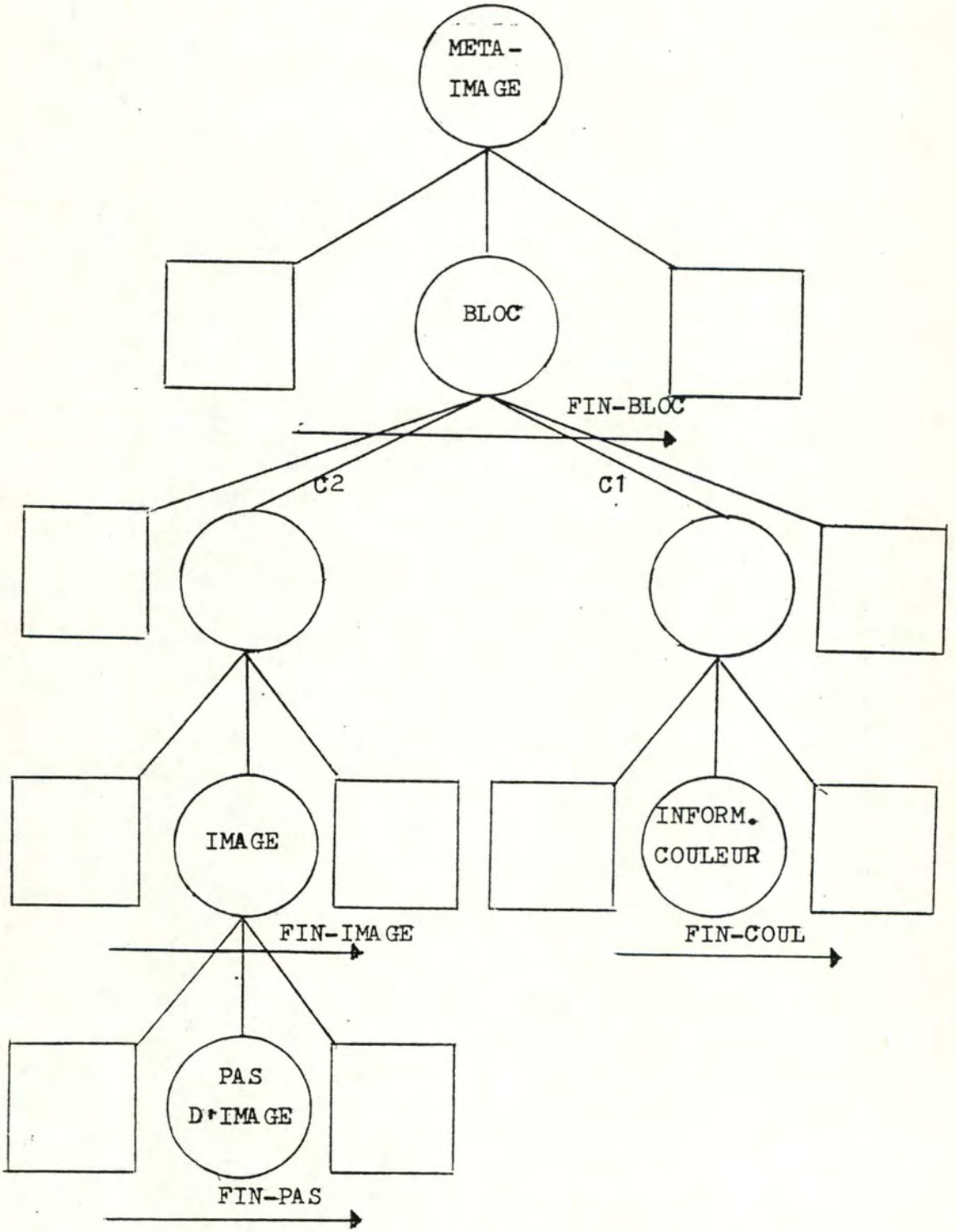
```
CODAGE :
     lire Image-stockée ;
     tant que pas Fin-image faire ;
          Zone-image = ZONE(coordonnée) ;
          selon Zone-image faire ;
               image    : ;
               menu     : C-MENU(coordonnée) ;
               palette  : C-PALETTE(coordonnée) ;
               pinceau  : C-PINCEAU(coordonnée) ;
          fin ;
     fin ;
fin ;
```

1.3.3.3. Décodage

Le décodage est analogue au codage et ne sera pas repris ici.

1.3.4. Module fonctionnel OPTIM

1.3.4.1. Structure des données



C1 : Informations-suivantes $\stackrel{?}{=}$ Description-couleurs

C2 : Informations-suivantes = Description-image

1.3.4.2. Module principal

```
OPTIM:
     Initialisation-globale;
     tant que pas Fin-opt;
         si Debut-image
           alors Init-nouvelle-image;
         Init-Pas-Image;
         tant que pas Fin-pas;
             si VALIDE(symbole)
               alors OPTIMISER(symbole);
         fin;
         SAUVER-PI-OPTIMISE;
     fin;
     Terminaison-globale;
fin;
```

1.3.4.3. Gestion des E/S

```
DIVISE:
      lire Image;
      si Information-couleur
        alors tant que pas Fin-couleur;
                  écrire Image;
                  lire Image;
              fin;
        sinon tant que pas Fin-symboles;
                  écrire Image;
                  lire Image;
              fin;
fin;
```

1.3.4.4. Gestion des Buffers

J'ai choisi la gestion des buffers parcequ'elle est non-triviale du fait de l'existence de deux symboles différents, les codes et les paramètres. Plusieurs choix ont été faits quant à l'utilisation des buffers:

3. la longueur du buffer est un multiple de la longueur d'un paramètre, et donc d'un code (1 paramètre = 2 codes).
4. chaque paramètre se trouve entièrement dans le buffer
5. le dernier élément lu est toujours gardé pour pouvoir satisfaire aux principes d'optimisation 4 et 5.

```
REMPLIR-BUFFER (Position);
     lire Image;
     Fin-buffer := faux;
     tant que pas (Fin-image ou Fin-buffer);
         Buffer(Position) = symbole;
         si Paramètre(symbole)
           alors lire Image;
                 Buffer(Position + 1) = symbole;
         Position = AUGMENTER(Position,symbole);
         Fin-buffer = EVALUER-FIN(Position,Max-Pos);
      fin;
fin;


EVALUER-FIN (Position,Max-Pos);
     retourner (Position >= Max-Pos);
fin;


AVANCER:
     si EVALUER-FIN(Pos-courante,Max-Pos)
       alors CHANGER-ELEMENT(Pos-courante,Min-Pos);
             REMPLIR-BUFFER(AUGMENTER(Min-Pos,Buffer(Min-Pos)));
             Pos-courante = Min-Pos;
     Pos-courante = AUGMENTER(Pos-courante,Buffer(Pos-courante));
fin;
```

1.3.4.5. Validation du pinceau

```
VALID-PINCEAU (symbole):
     si PINCEAU-PLUS(symbole)
       alors retourner (Etat-pinceau = Max-Pinceau);
       sinon retourner (Etat-pinceau = Min-Pinceau);
fin;
```

1.3.4.6. Optimisation de la palette

Dans le cas de l'optimisation de la palette, on suppose qu'il existe une structure qui peut contenir toutes les modifications apportées à la palette, une fonction PRED donnant le prédécesseur d'un élément de la structure et une fonction VIDE permettant de savoir si un élément existe ou non (indicateur du début de la structure). L'algorithme montré est général et ne contient par exemple pas le mode d'avancement dans la structure.

```
OPT-PALETTE (symbole):
     si ISBN(symbole)
       alors OPT-ISBN(symbole);
       sinon OPT-TEINTE(symbole);
fin;
```

```
OPT-TEINTE (symbole):
     si EXISTE(PREDECESSEUR(symbole))
       alors si CHOIX(symbole)
               alors OPT-CHOIX(symbole);
               sinon OPT-AVOIS(symbole);
       sinon ok;
fin;


OPT-CHOIX (symbole):
     si AVOISINAGE(PREDECESSEUR(symbole))
       alors ok;
       sinon si SELECTION(symbole)
               alors si SELECTION(PREDECESSEUR(symbole))
                       alors IDENTITE(symbole,
                                      PREDECESSEUR(symbole));
                             COMPRIMER;
                       sinon ok;
               sinon si SELECTION(PREDECESSEUR(symbole))
                       alors ECHANGER(symbole,
                                      PREDECESSEUR(symbole));
                             OPT-TEINTE(PREDECESSEUR(symbole));
                       sinon CHEMIN-OPT(symbole,
                                        PREDECESSEUR(symbole));
                             COMPRIMER;
fin;


OPT-AVOIS (symbole):
     si SELECTION(PREDECESSEUR(symbole))
       alors ok;
       sinon si AVOIS(PREDECESSEUR(symbole))
               alors CHEMIN-OPT(symbole,
                                PREDECESSEUR(symbole));
                     OPT-TEINTE(PREDECESSEUR(symbole));
               sinon IDENTITE(symbole,
                              PREDECESSEUR(symbole));
                     COMPRIMER;
fin;
```

Remarques : Le design qui vient d'être exposé ne donne qu'une idée générale du travail effectué, il ne permettrait probablement pas le codage de manière aisée. Mon propos n'était pas de permettre à quelqu'un de "refaire" le travail sur base de ce qui vient d'être dit, mais plutôt de montrer la méthodologie utilisé, phase par phase .

ANNEXE 2 : MANUEL D'UTILISATION DE MACCOI

2.1. Introduction générale

Le logiciel proposé permet de construire des images en utilisant le logiciel PICASSO, de les manipuler et de les stocker en vue d'une constitution d'une bibliothèque d'images. Il permet en outre de compacter et ensuite d'optimiser les images stockées afin d'occuper un minimum de place sur disque.
Comme le logiciel est formé de 3 parties différentes correspondant à 3 programmes, on donnera successivement leur mode d'utilisation.

Afin de pouvoir travailler avec le logiciel proposé, il faut faire démarrer le système avel le logiciel RUN dont l'utilisation est expliqué à la fin de ce manuel.

2.2. Le programme MIDESS

2.2.1. Introduction

MIDESS permet de créer des images en utilisant le logiciel PICASSO, d'afficher des imaes pour les modifier par après et de superposer des images au dessin affiché sur l'écran graphique.
En plus, il y a moyen de renommer des images ou bien d'en détruire. Une fonction de stockage permet de stocker une image sur fichier et relance le système. Enfin, une fonction de sortie permet de terminer la session de travail.

2.2.2. Appel du programme

MIDESS est appelé par le ligne de commande suivante:
:F4:MIDESS ‹RET›

2.2.3. Utilisation de MIDESS

L'utilisateur devra se patienter quelques instants avant que le système soit initialisé. MIDESS lui demandera ensuite un nom de catalogue (tel que F6) nécessaire pour y mettre un fichier de travail. Ce nom de catalogue doit être le même que celui dans lequel on va sauver l'image finale.

Après que l'utilisateur ait donné sa réponse, le programme lui présente le menu principal et attend une réponse sous forme d'un chiffre, lequel désigne le traitement à effectuer. MIDESS demandera toujours confirmation et l'utilisateur n'est pas tenu à mettre un ‹RET› après chaque entrée, MIDESS s'en occupe.

2.2.3.1. Crèer

Un message signale l'utilisation de cette fonction. L'utilisateur peut alors crèer une image en utilisant PICASSO.

2.2.3.2. Afficher

Cette fonction permet d'afficher une image sur un ècran vide. MIDESS efface donc l'ècran avant de demander le nom de l'image à afficher.

2.2.3.3. Renommer

La fonction demande d'abord le nom de l'image à renommer, puis le nouveau nom.

2.2.3.4. Dètruire

Cette fonction, qui permet d'enlever une image de la bibliothèque de l'utilisateur, demande le nom de l'image à dètruire.

2.2.3.5. Superposer

Cette fonction suppose qu'une image existe dèja sur l'ècran et permet de superposer une image stockèe à l'ècran. L'image à superposer est dessinèe au-dessus de celle èventuellement prèsente sur l'ècran; il n'y a pas de mèlange de couleurs. Par-contre, le fait que l'ècran peut être non vide peut avoir des consèquences lors de l'excution de certaines fonctions de la famille COULEUR, la couleur du fond ou les contours des èlèments ayant changès.

2.2.3.6. Stocker

Permet de sauver le travail effectuè jusqu'à ce moment et de recommencer à travailler à partir d'un ècran vide. On ne peut donc pas sauver un rèsultat intermèdiaire et continuer à dessiner; pour ce faire, il faut sauver et puis rèafficher l'image en question. La fonction demande un nom de fichier à appliquer à l'image et n'accepte pour le moment qu'un nom n'existant pas encore.

2.2.3.7. Terminer

Fonction qui permet d'arrêter la session de travail. Elle permet de sauver l'image finale en procédant de même manière que la fonction Stocker, mais sort de MIDESS par après.

2.2.4. Recouvrement d'erreurs

1. un nom de fichier vide conduit à l'annulation de l'action à effectuer.
2. La structure des noms de fichier doit être respectè
3. les chiffres entrées doivent respecter les bornes du menu, donc au plus 0 <= chiffre <= 9.
4. la confirmation se fait par les lettres "o/O/n/N"

Chaque erreur produit un message. L'utilisateur est prié de le lire, de taper <RET> ce qui a pour effet de signaler la prise en compte de l'erreur et de répondre de nouveau à la question.

2.3. Le programme CODE

2.3.1. Introduction

Dans le cas où l'utilisateur dispose de peu de place pour stocker ses images, il peut être soucieux de les garder sous une forme compacte. Le programme CODE permet à la fois de coder et de décoder une image (car MIDESS n'accepte que les images non codées).

2.3.2. Appel de CODE

2.3.2.1. Paramètres d'entrée

L'invocation du programme diffère des autres appels par le fait qu'elle nécessite des paramètres sur la ligne de commande. CODE n'a aucune autre interaction avec l'utilisateur.
Outre le nom du programme, il faut fournir deux paramètres à CODE:

1. un code indiquant si l'on veut coder ("C") ou décoder ("D") une image
2. le nom de l'image à coder/décoder

Voici un exemple d'une ligne de commande :

```
:F4:CODE C :F6:DESSIN.NEW
```

2.3.2.2. Résultat

Le résultat dépend de l'action qu'on vient d'effectuer:

1. on veut coder une image existante (p.ex. :F6:DESSIN.NEW)
   Le résultat sera une image codée de nom :F6:DESSIN.COD
2. on veut décoder une image (p.ex. :F6:DESSIN.COD)
   Le résultat sera une image décodée de nom :F6:DESSIN.DEC

L'image d'origine n'est pas perdue.

2.3.3. Recouvrement d'erreurs

Le recouvrement d'erreurs est plus primitif que pour les autres programmes. Si le nom de l'image est invalide, aucune action sera exécutée.
Cas d'une image physiquement valide, mais pas logiquement.

1. on veut encoder une image codée.
   Le résultat sera un fichier..COD de contenu invalide, l'image sera ainsi perdue, donc ATTENTION.
2. on veut décoder une image décodée.
   Le résultat sera la même image, non altérée. Cette faute conduit donc à une action vide.

2.4. Le programme OPTIM

2.4.1. Introduction

Souvent, en travaillant, on hésite sur le choix d'une couleur ou d'une action à effectuer, on choisit une fonction pour finalement en utiliser une autre, on donne plus de paramètres à une fonction qu'il n'en faut, etc.
Toutes ces actions n'apportent rien au produit final et peuvent dès lors être éliminées.
OPTIM réalise ceci et produit une image où ne subsiste que le stricte nécessaire à la production d'une image. Entre autres, l'optimisation enlève la grille, qui ne sert qu'à cadrer les paramètres sur l'écran.

2.4.2. Appel du programme

2.4.2.1. Paramètre d'entrée

Le seul paramètre d'entrée est le nom de l'image à optimiser. La ligne de commande est la suivante:

```
:F4:OPTIM
```

OPTIM demandera alors le nom à l'utilisateur, de manière identique que dans MIDESS.

2.4.2.2. Résultat

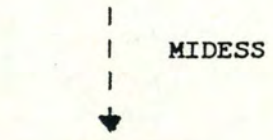
OPTIM produit une image optimisée dont l'extension sera ".OPT". Cette image est évidemment codée est doit d'abord être décodée avant de pouvoir être utilisée dans MIDESS.

2.4.3. Recouvrement d'erreurs

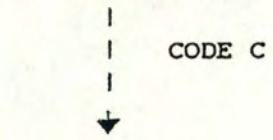
id. MIDESS

2.5. Conclusion

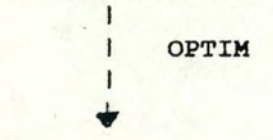
On vient de montrer comment utiliser le logiciel propose pour constituer une bibliotheque d'images nécessitant un minimum de place de stockage.
Le cheminement a été le suivant:



:F4:DESSIN.NEW



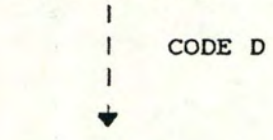
:F4:DESSIN.COD



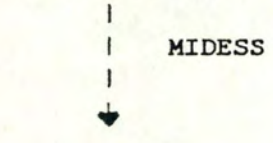
:F4:DESSIN.OPT

On peut maintenant détruire les fichiers :F4:DESSIN.NEW et :F4:DESSIN.COD. Pour réutiliser l'image avec MIDESS, il va falloir suivre le chemin inverse:

:F4:DESSIN.OPT



:F4:DESSIN.DEC



2.6. Démarrage du système

Pour allumer la machine, le mieux qu'on puisse faire est de demander à quelqu'un du laboratoire.
Si la machine est allumée, des astérisques apparaissent, il suffit de taper un "U" (majuscule) pour pouvoir travailler.
Il faut charger le programme de gestion maintenant en tapant la ligne de commande suivante:

```
BL PIREN RUN <RET>
```

Le logiciel appelé demandera la date et les catalogues à utiliser. A moins que l'utilisateur veuille utiliser ses propres catalogues, la procédure est la suivante:

```
"catalogue?"    F4=US:SPANAL.OBJ <RET>
"catalogue?"    F6=US:SPANAL.FLS <RET>
"catalogue?"    <RET>
"work?          <RET>
```

Le logiciel va maintenant donner la main à l'utilisateur en mettant un signe ">" en première colonne. MIDESS/CODE/OPTIM peuvent alors être appelés.

2.7. Structure d'un nom d'image

Elle est la suivante:

```
:nom-catalogue:nom-propre.extension

      avec nom-catalogue = 2 caractères (A..Z, 0..9)
                           (p.ex. FF, A5, F4)
           nom-propre    = 6 caractères maximum
                           dont le premier est alphabétique
           extension     = 3 caractères maximum
```

Exemples de noms d'image:

1. valides
   - :F4:DESSIN.NEW
   - :F6:A1223.001
   - :XZ:A.Z80
2. invalides
   - :12:DESSIN.COD
   - nom-catalogues invalide :F4:123.GO
   - nom-propre " :F4:A
   - exension " :F4:A. "
   - "

2.8. Exemple d'une session de travail avec MIDESS

On va illustrer le fonctionnement de MIDESS par un exemple.
Convenons qu'on va représenter les messages du programme en minuscules entre guillemets et les réponses de l'utilisateur en majuscules.

```
:F4:MIDESS <RET>
    "entrez le nom du catalogue de travail s.v.p : "  P6
             "ok ? (o/n) : "  o
- le menu principal va s'afficher -
    "vous pouvez
         1) dessiner
         2) afficher
         3) ..
         7) terminer
         entrez un chiffre entre 1 et 7 :"  1
    " ok? (o/n) :" o
    "vous pouvez dessiner maintenant"
```

on commence par dessiner et on arrête par STOP (fonction de la famille SORTIE de PICASSO).
Le menu principal va réapparaître, on entre 7 cette fois-ci:

```
   "voulez-vous
      1) sauvez le resultat
      2) ne rien changer
      entrez 1 ou 2 :"     1
  - MIDESS demande alors le nom de l'image -
   "entrez le nom de l'image à utiliser :" :F4:DESSIN.NEW
   "ok ? (o/n) " o
```

On sort du programme.