50376
1986
153

N° d'ordre : 385

UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE FLANDRES ARTOIS

THÈSE

présentée à

L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE FLANDRES ARTOIS

pour obtenir le titre de

DOCTEUR INGENIEUR

par

Yahya SLIMANI

STRUCTURES DE DONNEES ET LANGAGES NON-PROCEDURAUX EN INFORMATIQUE GRAPHIQUE

Thèse soutenue le 7 février 1986 devant la Commission d'Examen

Membres du Jury :	V. CORDONNIER	Président
	M. MERIAUX	Rapporteur
	H. ROTHLISBERGER	Examinateur
	P. LECOUFFE	Examinateur
	P. DAUBRESSE	Examinateur

P R O F E S S E U R S CLASSE EXCEPTIONNELLE

M. CONSTANT Eugène	I.E.E.A.
M. FOURET René	Physique
M. GABILLARD Robert	I.E.E.A.
M. MONTREUIL Jean	Biologie
M. PARREAU Michel	Mathématiques
M. TRIDOT Gabriel	Chimie
M. VIVIER Emile	Biologie
M. WERTHEIMER Raymond	Physique

P R O F E S S E U R S 1ère classe

M. BACCHUS Pierre	Mathématiques
M. BEAUFILS Jean-Pierre (dét.)	Chimie
M. BIAYS Pierre	G.A.S.
M. BILLARD Jean (dét.)	Physique
M. BOILLY Bénoni	Biologie
M. BOIS Pierre	Mathématiques
M. BONNELLE Jean-Pierre	Chimie
M. BOUGHON Pierre	Mathématiques
M. BOURIQUET Robert	Biologie
M. BREZINSKI Claude	I.E.E.A.
M. CELET Paul	Sciences de la Terre
M. CHAMLEY Hervé	Biologie
M. COEURE Gérard	Mathématiques
M. CORDONNIER Vincent	I.E.E.A.
M. DEBOURSE Jean-Pierre	S.E.S.
M. DYMENT Arthur	Mathématiques

PROFESSEURS 1ère classe (suite)

M. ESCAIG Bertrand	Physique
M. FAURE Robert	Mathématiques
M. FOCT Jacques	Chimie
M. GRANELLE Jean-Jacques	S.E.S.
M. GRUSON Laurent	Mathématiques
M. GUILLAUME Jean	Biologie
M. HECTOR Joseph	Mathématiques
M. LABLACHE COMBIER Alain	Chimie
M. LACOSTE Louis	Biologie
M. LAVEINE Jean Pierre	Sciences de la Terre
M. LEHMANN Daniel	Mathématiques
Mme LENOBLE Jacqueline	Physique
M. LHOMME Jean	Chimie
M. LOMBARD Jacques	S.E.S.
M. LOUCHEUX Claude	Chimie
M. LUCQUIN Michel	Chimie
M. MIGEON Michel Recteur à Grenoble	E.U.D.I.L.
M. MIGNOT Fulbert (dét.)	Mathématiques
M. PAQUET Jacques	Sciences de la Terre
M. PROUVOST Jean	Sciences de la Terre
M. ROUSSEAU Jean-Paul	Biologie
M. SALMER Georges	I.E.E.A.
M. SEGUIER Guy	I.E.E.A.
M. SIMON Michel	S.E.S.
M. STANKIEWICZ François	S.E.S.
M. TILLIEU Jacques	Physique
M. VIDAL Pierre	I.E.E.A.
M. ZEYTOUNIAN Radyadour	Mathématiques

P R O F E S S E U R S 2ème classe

M. ANTOINE Philippe	Mathématiques (Calais)
M. BART André	Biologie
Mme BATTIAU Yvonne	Géographie
M. BEGUIN Paul	Mathématiques
M. BELLET Jean	Physique
M. BERZIN Robert	Mathématiques
M. BKOUCHE Rudolphe	Mathématiques
M. BODARD Marcel	Biologie
M. BOSQ Denis	Mathématiques
M. BRASSELET Jean-Paul	Mathématiques
M. BRUYELLE Pierre	Géographie
M. CAPURON Alfred	Biologie
M. CARREZ Christian	I.E.E.A.
M. CAYATTE Jean-Louis	S.E.S.
M. CHAPOTON Alain	C.U.E.E.P.
M. COQUERY Jean-Marie	Biologie
Mme CORSIN Paule	Sciences de la Terre
M. CORTOIS Jean	Physique
M. COUTURIER Daniel	Chimie
M. CROSNIER Yves	I.E.E.A.
M. CURGY Jean-Jacques	Biologie
Mle DACHARRY Monique	Géographie
M. DAUCHET Max	I.E.E.A.
M. DEBRABANT Pierre	E.U.D.I.L.
M. DEGAUQUE Pierre	I.E.E.A.
M. DELORME Pierre	Biologie
M. DELORME Robert	S.E.S.
M. DE MASSON D'AUTUME Antoine	S.E.S.
M. DEMUNTER Paul	C.U.E.E.P.

PROFESSEURS 2ème classe (Suite 1)

M. DENEL Jacques	I.E.E.A.
M. DE PARIS Jean-Claude	Mathématiques (Calais)
Mle DESSAUX Odile	Chimie
M. DEVRAINNE Pierre	Chimie
M. DHAINAUT André	Biologie
Mme DHAINAUT Nicole	Biologie
M. DORMARD Serge	S.E.S.
M. DOUKHAN Jean-Claude	E.U.D.I.L.
M. DUBOIS Henri	Physique
M. DUBRULLE Alain	Physique (Calais)
M. DUBUS Jean-Paul	I.E.E.A.
M. FAKIR Sabah	Mathématiques
M. FONTAINE Hubert	Physique
M. FOUQUART Yves	Physique
M. FRONTIER Serge	Biologie
M. GAMBLIN André	G.A.S.
M. GLORIEUX Pierre	Physique
M. GOBLOT Rémi	Mathématiques
M. GOSSELIN Gabriel (dét.)	S.E.S.
M. GOUDMAND Pierre	Chimie
M. GREGORY Pierre	I.P.A.
M. GREMY Jean-Paul	S.E.S.
M. GREVET Patrice	S.E.S.
M. GUILBAULT Pierre	Biologie
M. HENRY Jean-Pierre	E.U.D.I.L.
M. HERMAN Maurice	Physique
M. JACOB Gérard	I.E.E.A.
M. JACOB Pierre	Mathématiques
M. JEAN Raymond	Biologie
M. JOFFRE Patrick	I.P.A.

PROFESSEURS 2ème classe (suite 2)

M. JOURNEL Gérard	E.U.D.I.L.
M. KREMBEL Jean	Biologie
M. LANGRAND Claude	Mathématiques
M. LATTEUX Michel	I.E.E.A.
Mme LECLERCQ Ginette	Chimie
M. LEFEVRE Christian	Sciences de la Terre
Mle LEGRAND Denise	Mathématiques
Mle LEGRAND Solange	Mathématiques (Calais)
Mme LEHMANN Josiane	Mathématiques
M. LEMAIRE Jean	Physique
M. LHENAFF René	Géographie
M. LOCQUENEUX Robert	Physique
M. LOSFELD Joseph	C.U.E.E.P.
M. LOUAGE Francis(dét.)	E.U.D.I.L.
M. MACKE Bruno	Physique
M. MAIZIERES Christian	I.E.E.A.
M. MESSELYN Jean	Physique
M. MESSERLIN Patrick	S.E.S.
M. MONTEL Marc	Physique
Mme MOUNIER Yvonne	Biologie
M. PARSY Fernand	Mathématiques
Mle PAUPARDIN Colette	Biologie
M. PERROT Pierre	Chimie
M. PERTUZON Emile	Biologie
M. PONSOLLE Louis	Chimie
M. PORCHET Maurice	Biologie
M. POVY Lucien	E.U.D.I.L.
M. RACZY Ladislas	I.E.E.A.
M. RAOULT Jean François	Sciences de la Terre
M. RICHARD Alain	Biologie

PROFESSEURS 2ème Classe (suite 3)

M. RIETSCH François	E.U.D.I.L.
M. ROBINET Jean-Claude	E.U.D.I.L.
M. ROGALSKI Marc	Mathématiques
M. ROY Jean-Claude	Biologie
M. SCHAMPS Joël	Physique
Mme SCHWARZBACH Yvette	Mathématiques
M. SLIWA Henri	Chimie
M. SOMME Jean	G.A.S.
Mle SPIK Geneviève	Biologie
M. STAROSWIECKI Marcel	E.U.D.I.L.
M. STERBOUL François	E.U.D.I.L.
M. TAILLIEZ Roger	Institut Agricole
Mme TJOTTA Jacqueline (dét.)	Mathématiques
M. TOULOTTE Jean-Marc	I.E.E.A.
M. TURRELL Georges	Chimie
M. VANDORPE Bernard	E.U.D.I.L.
M. VAST Pierre	Chimie
M. VERBERT André	Biologie
M. VERNET Philippe	Biologie
M. WALLART Francis	Chimie
M. WARTEL Michel	Chimie
M. WATERLOT Michel	Sciences de la Terre
Mme ZINN JUSTIN Nicole	Mathématiques

CHARGES DE COURS

M. ADAM Michel	S.E.S.

CHARGES DE CONFERENCES

M. BAFCOP Joël	I.P.A.
M. DUVEAU Jacques	S.E.S.
M. HOFLACK Jean	I.P.A.
M. LATOUCHE Serge	S.E.S.
M. MALAUSSENA DE PERNO Jean-Louis	S.E.S.
M. NAVARRE Christian	I.P.A.
M. OPIGEZ Philippe	S.E.S.

REMERCIEMENTS

Je tiens à remercier Monsieur V. CORDONNIER, Professeur à l'Université de LILLE I, pour l'honneur qu'il me fait en acceptant de présider ce jury.

Qu'il trouve ici l'expression de ma profonde gratitude pour m'avoir accepté au sein de son laboratoire et pour m'avoir permis à travers ce sujet de découvrir le domaine passionnant du graphique. Je n'omettrai pas de le remercier également pour les conseils et encouragements qu'il n'a cessé de me prodiguer tout au long de ces travaux malgré ses nombreuses occupations.

Que Monsieur M. MERIAUX, Chargé de Recherches au C.N.R.S., trouve ici toute ma reconnaissance pour avoir suivi et guidé tous les développements de mes recherches. Grâce à sa compétence et à son dynamisme dans le graphique, nous avons eu beaucoup de discussions fructueuses sans lesquelles un tel travail ne se serait pas concrétisé.

Qu'il trouve ici mes vifs remerciements pour la célérité avec laquelle il a "épluché" les différentes versions de cette thèse, ainsi que pour ses remarques qui ont permis d'aboutir à cette version finale.

Monsieur H. ROTHILSBERGER, Professeur à l'Ecole d'Ingénieurs de l'Etat de VAUD à YVERDON-LES-BAINS (Suisse), s'est intéressé dès le début aux travaux que je présente ici, malgré la distance nous séparant.

Qu'il trouve ici toute ma reconnaissance d'avoir bien voulu participer au jury et accepté d'en être examinateur. Je le remercie vivement d'avoir su m'encourager par une claivoyance constante et de m'avoir communiqué une vision précise des problèmes importants en graphique. Ses critiques constructives ont par ailleurs largement contribué à améliorer le texte final.

Je remercie également Monsieur P. LECOUFFE, Maître-Assistant à l'I.U.T. de Lille et Monsieur P. DAUBRESSE, Ingénieur - Directeur de la SODIA d'avoir bien voulu accepter de juger ce travail.

.../...

Je tiens à adresser également mes remerciements à Madame C. LAVERDISSE et Monsieur H. GLANC qui ont assuré avec gentillesse et compétence la réalisation matérielle de ce document.

Si une thèse est en général le fruit de toute une équipe, je suis particulièrement fier de dire que celle-ci est également une affaire de famille. En effet, comment ne pas faire une place dans ces pages, à ma femme, mes enfants et ma famille qui ont su, malgré la distance qui nous séparait, supporter mon absence et me communiquer le courage dont j'ai eu besoin pour atteindre le but final.

A la mémoire de mon père

A ma mère

A ma femme

A mes enfants

S O M M A I R E

INTRODUCTION GENERALE

L'avènement de l'image dans le monde informatique et l'intérêt toujours croissant qui lui est accordé, découlent essentiellement de la quantité d'informations qu'elle transmet sous une forme synthétique ou qu'elle suggère, sans nécessiter de la part de son observateur aucune connaissance ni "culture" particulière. Cet intérêret est de plus en plus accru par les divers secteurs que recouvre l'informatique graphique (industriel, économique, social, culturel, etc...) grâce à la plus grande maîtrise que les utilisateurs en ont, ainsi que par l'évolution de la technologie, mais surtout grâce aux améliorations qu'elle a apporté à l'interaction homme/machine.

D'un autre côté, l'évolution de l'informatique peut se mesurer à celle des langages de programmation [WEG 76], depuis le binaire pur jusqu'au langage ADA, le dernier en date parmi les plus connus. Mais cette évolution des langages de programmation risque de connaître un grand tournant par l'utilisation et la généralisation des langages dits de l'intelligence artificielle (supports des architectures de la 5ème génération), qui font une entrée remarquable et remarquée en informatique. Cette entrée est plus exactement une redécouverte et un intérêt sans cesse grandissant à ces langages, dont le précurseur (LISP) existe depuis plus de 20 ans. Ces types de langages se différencient fondamentalement des langages de programmation usuels, en introduisant un nouvel esprit de programmation qui est celui de la programmation non-procédurale [BAC 78, COL 83, SLI 84, SLI 85].

C'est ainsi que nous nous sommes proposés, à travers cette thèse, de rechercher ce que pourraient apporter de tels langages (et plus exactement leurs concepts) au domaine graphique, notamment en ce qui concerne l'expression d'algorithmes graphiques, et la représentation des images. Il va de soi que le but d'une telle étude n'est pas tant l'amélioration de la qualité des images sur ordinateur, mais plutôt l'exploration des caractéristiques qu'offrent de tels langages pour le graphique.

Cette thèse permet de concrétiser une telle étude en couvrant les aspects suivants :

i) Un aspect structures de données où après une présentation critique des différents modèles de structures de données, nous mettons en évidence

les relations existantes entre structures de données et systèmes graphiques. Nous présenterons par la suite, sur la base de nos expérimentations, un nouveau modèle de structure de donnée suceptible d'être utilisé pour représenter les objets graphiques, à savoir la structure de liste.

ii) Un aspect langages et algorithmique par l'utilisation d'un langage applicatif (LE-LISP) pour l'expression d'algorithmes graphiques. Nous montrerons comment un des concepts fondamentaux de ces langages (la récursivité) peut contribuer à une expression naturelle d'algorithmes graphiques.

Le premier chapitre de ce travail est consacré essentiellement à une présentation générale des systèmes graphiques. Beaucoup d'ouvrages et d'articles ayant été écrits sur cet aspect du graphique, nous nous contenterons d'en faire un bref rappel bibliographique pour le lecteur intéressé. Nous insisterons pour les besoins de notre étude sur les aspects images et langages graphiques.

Après un aperçu sur les problèmes posés par les systèmes graphiques en matière de structures de données, le deuxième chapitre essaye de mettre en évidence les relations existantes entre systèmes graphiques et structures de données.

Le troisième chapitre est consacré aux langages de programmation utilisables en graphique. Après une analyse critique des langages procéduraux qui représentent plus de 95% des langages utilisables actuellement, nous présentons les concepts fondamentaux des langages non procéduraux. Pour les besoins de notre étude, cette présentation est faite avec trois langages représentatifs de la programmation non procédurale (LISP, PROLOG, SMALLTALK). Nous terminerons ce chapitre par la spécification du type de langage choisi pour les besoins de notre expérimentation (LE-LISP).

Le quatrième chapitre est consacré essentiellement à la présentation conceptuelle du langage choisi, de l'environnement de programmation ainsi qu'à l'expérimentation réalisée en faisant ressortir les difficultés d'implémentation. Nous terminerons ce chapitre par une analyse-bilan de cette étude ainsi que quelques prolongements possibles de ce travail.

CHAPITRE 1 : GENERALITES SUR LES SYSTEMES GRAPHIQUES

CHAPITRE I

GENERALITES SUR LES SYSTEMES GRAPHIQUES

****
****
****

1.1. Introduction

Les jeux vidéo ont constitué et constituent d'ailleurs actuellement avec le développement de la micro-informatique, la première utilisation domestique à grande échelle de l'informatique graphique, c'est-à-dire la création et la manipulation d'images à l'aide d'un calculateur. Un autre exemple familier de cette forme est la télévision. On estime même que dans un temps relativement proche, une grande partie des images de télévision, de cinéma, sera composée d'images de synthèse, sans compter leur expansion dans des domaines tels que la conception assistée par ordinateur (C.A.O.), les simulateurs de vols, les dessins animés, etc...

Pour situer le développement dans le temps, on peut dire que les années 60 ont représenté les années de pointe sur la recherche des calculateurs graphiques (le système SKETCHPAD conçu en 1963 par I.E. SUTHERLAND [MAC 84-a] est considéré comme étant l'ancêtre des systèmes graphiques), alors que les années 70 ont représenté la décade au cours de laquelle les recherches ont commencé à donner des résultats. Actuellement, on peut considérer qu'au vu des résultats obtenus, des progrès technologiques et de la diversité sans cesse croissante des champs d'applications, l'utilisation des calculateurs pour la production d'images sera de plus en plus accrue. L'avantage d'un tel développement suscite de nombreuses réflexions sur les concepts classiques de l'informatique (architectures, structures de données, algorithmique, programmation, langages, etc...), qui s'avèrent de plus en plus inadaptés pour la résolution de problèmes graphiques. C'est dans ce cadre là, que notre travail s'intègre, en menant une réflexion sur les problèmes de structures de données et de langages.

1.2. Qu'est-ce qu'un système graphique

On peut définir un système graphique interactif, au sens général, comme étant un système permettant la création, le stockage et la manipulation de modèles d'objets ainsi que leurs images à travers un calculateur. Ceci implique une intéraction constante entre l'utilisateur et le calculateur. Un tel système utilise comme puissant outil de communication l'image. En effet, l'image est le moyen de communication le plus naturel et le plus efficace car elle

véhicule une importante quantité d'informations sous forme synthétique, compréhensible par une grande population (Fig. 1.1.)

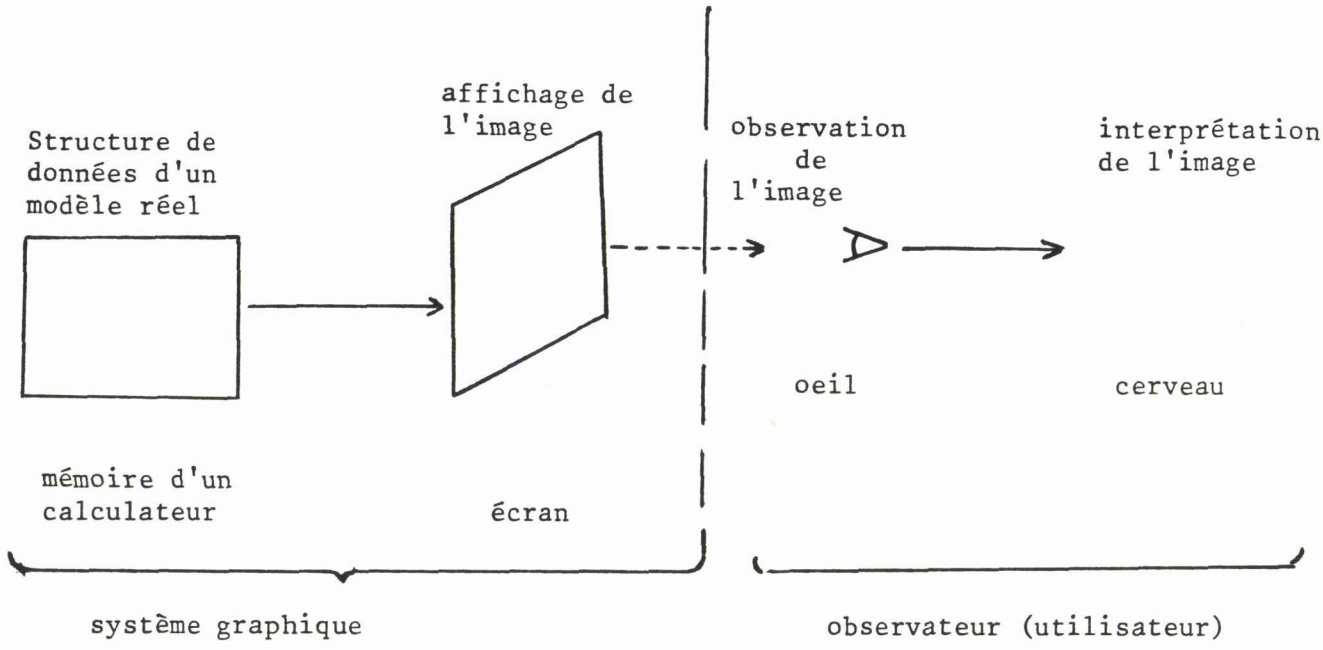


Fig. 1.1. Les différentes représentations d'une image

1.3. Les Ecrans

La littérature étant assez abondante dans la présentation de ces éléments matériels des systèmes graphiques, le lecteur intéressé pourra consulter les quelques références bibliographiques suivantes [MOR 76, LUC 77, MER 79, NEW 79, ROT 80, FOL 82, MAR 82]. On notera seulement que de nouveaux supports de visualisation sont attendus. On signale déjà la possibilité de disposer d'écrans avec une meilleure définition (1024 * 1024, bientôt 2048 * 2048 points). Il est tout à fait probable que des technologies différentes de la vidéo seront maîtrisées et fourniront de nouvelles possibilités d'expression [LUC 84].

1.4. L'Image

1.4.1. Introduction

La phase finale de tout système graphique étant la production d'images, il s'avère indispensable d'étudier la notion, à la fois simple et complexe,

d'image. Rappelons tout d'abord que l'image constitue le moyen de communication le plus naturel et le plus efficace car elle véhicule d'une part un nombre considérable d'informations sous forme synthétique et d'autre part elle est compréhensible par une large classe de personnes en ce fait qu'elle utilise comme puissant outil d'analyse le sens de la vision.

1.4.2. Définitions

Nous proposons dans ce paragraphe de définir la notion d'image sous trois aspects :

i) informel

ii) fonctionnel

iii) logique

Cette présentation, outre son aspect d'essai de formalisation, montre combien la notion d'image est mal appréhendée de part sa complexité et devra faire l'objet de formalisation car elle constitue l'objet de base de tout système graphique [FU-82, PRE 82, INR 83].

1.4.2.1. Aspect informel [PAR 82]

Une image peut être définie comme un descripteur d'une structure contenant deux types d'informations :

i) des informations liées à la structure de l'image, c'est-à-dire les éléments qui la composent

ii) des informations liées à la géométrie et à la visualisation, comme par exemple la forme des éléments, la couleur, la texture, etc...

$I = \{S, A\}$

$I \rightarrow$ objet image

$S \rightarrow$ information structure

$A \rightarrow$ information attributs graphiques.

Cet aspect permet d'avoir une description structurelle de l'image. C'est une description classique [NEW 79], car elle correspond au mode de pensée de

de l'utilisateur. Elle représente un moyen très naturel de concevoir la description des objets à visualiser. Par exemple une voiture est composée d'une partie mécanique et d'une partie carrosserie. La carrosserie est composée d'une coque, de portes, de capot, d'ailes, etc... (fig. 1.2.)

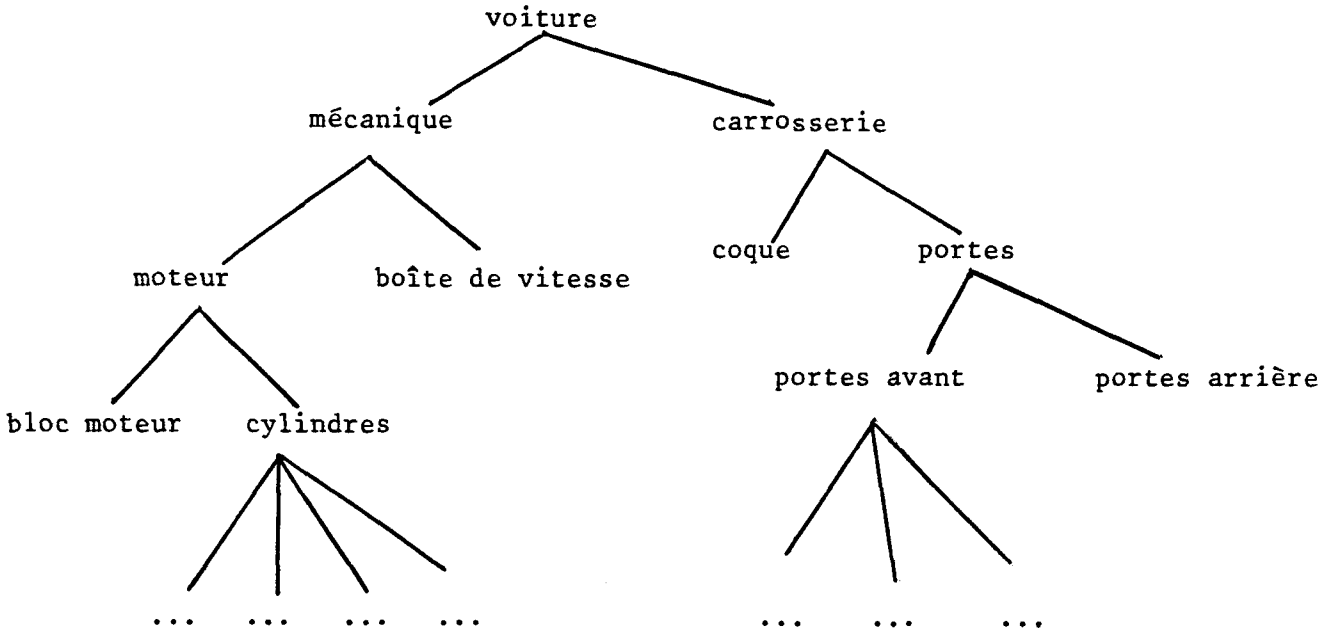


Fig. 1.2. Exemple de structure d'un objet-image

1.4.2.2. Aspect fonctionnel [INR 83]

De façon plus formelle on peut définir une image comme suit :

$A_i = F(C_i)$

ou

A_i ← attributs de l'image i

F ← fonction attribut

C_i ← coordonnées de l'image i

Ceci nous permet de définir une image d'un point de vue fonctionnel comme une fonction qui associe à tout point d'un plan un attribut qui est un ensemble de valeurs. La combinaison de fonctions attribut permet donc d'obtenir une description fonctionnelle d'une image donnée.

Exemples :

- Choix d'une couleur de pixel à l'aide d'une table de fausses couleurs :

  C (x,y) : = F (x,y)

- Fonctions de tranformations de la forme :

  I (x,y) : = J (X(x,y), Y (x,y))

  parmi lesquelles on trouve les transformations géométriques dont la plus couramment mise en oeuvre est la translation où

  X (x,y,) = x + dx et Y (x,y) = y + dy

1.4.2.3. Aspect logique [FU-82, PRE 82]

Au lieu de considérer les images comme des tableaux de pixels à deux dimensions, on peut les considérer à un niveau logique.

A ce niveau on définira une image comme suit :

i) un ensemble d'objets : ϕ

ii) un ensemble de relations entre ces objets : R

On crée ainsi une structure de donnée hiérarchisée de l'image.

Exemple :

ϕ_i = ((type,valt),(nom1,valn),(x_1,x_{i_1}),(x_2,x_{i_2}),...,$(x_n x_{i_n})$,(a_1,a_{i_1}),...,(a_k,a_{i_k}), (R,(nom2,valn)))

où

type → catégorie de l'objet (pixel, segment, carré, etc...)

nom1 → identification de l'objet

x_1,... x_n → coordonnées spatiales de l'objet

a_1,... a_k → attributs de l'objet

R → type de relation(appartenance, connectivité, etc...)

nom 2 → objet lié à l'objet courant (nom1) par la relation R.

Définition d'un pixel

((type, pixel), (x,x_i), (y,y_i), $(a,f(x_i,y_i))$, (app, (nom, seg_i)))

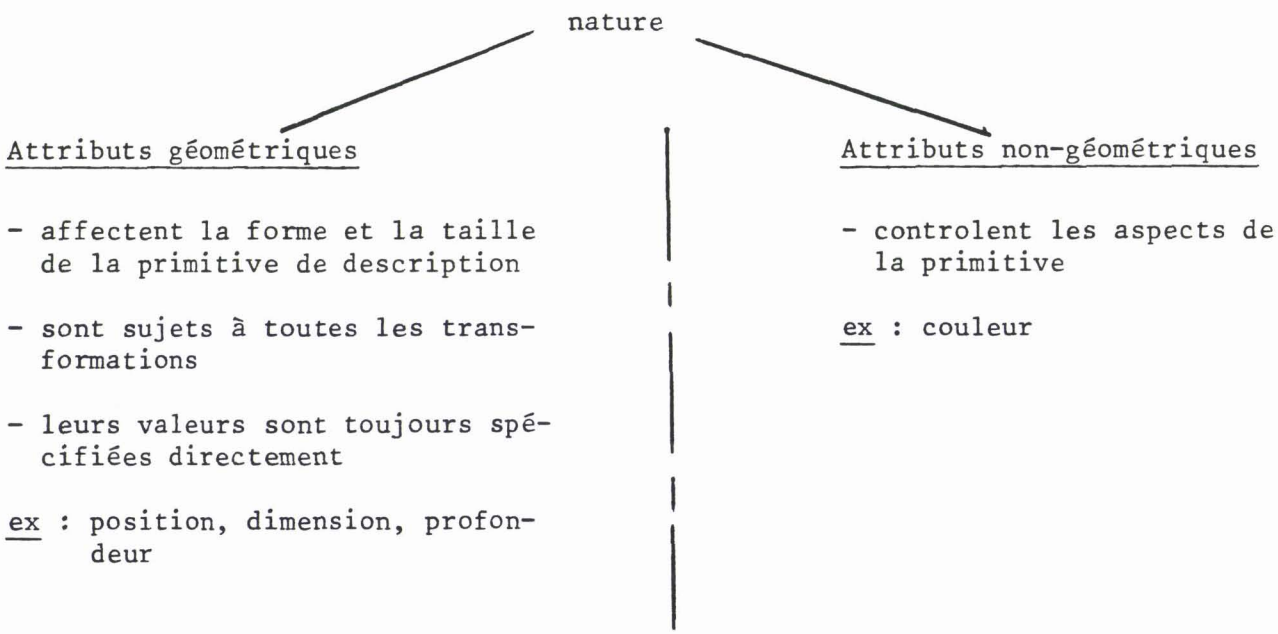
1.4.3. Informations véhiculées par une image

Les images du monde réel contiennent un grand nombre d'informations complexes telles que la texture, la couleur, l'ombrage, etc..., qui nécessitent des techniques de calcul assez puissantes (parallélisme, pipe-line, etc...).

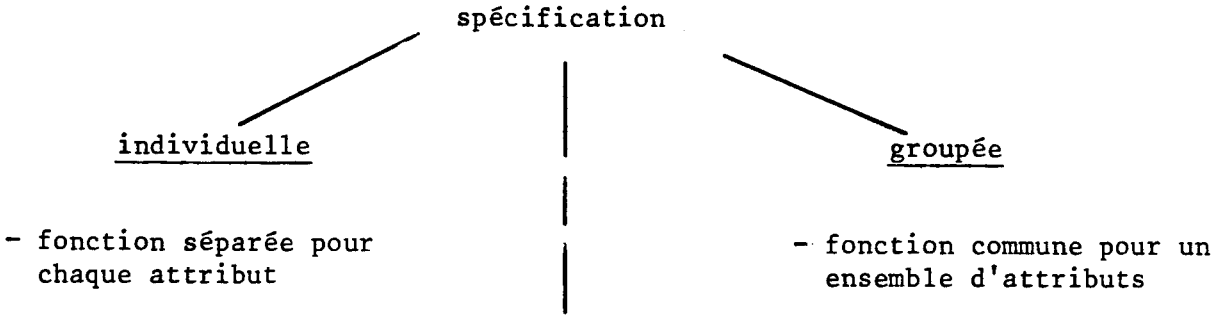
D'autre part une image comporte un nombre important d'informations pour quelles soient toutes utiles. En fait cette notion d'information utile dépend de l'application utilisant comme objet d'analyse, l'image, et influe plus ou moins sur la visualisation (choix des informations à visualiser).

D'une manière générale on peut classer les attributs d'une image sous différents aspects [END 84-a]. On distingue ainsi cinq aspects pouvant qualifier un attribut suivant la norme internationale GKS.

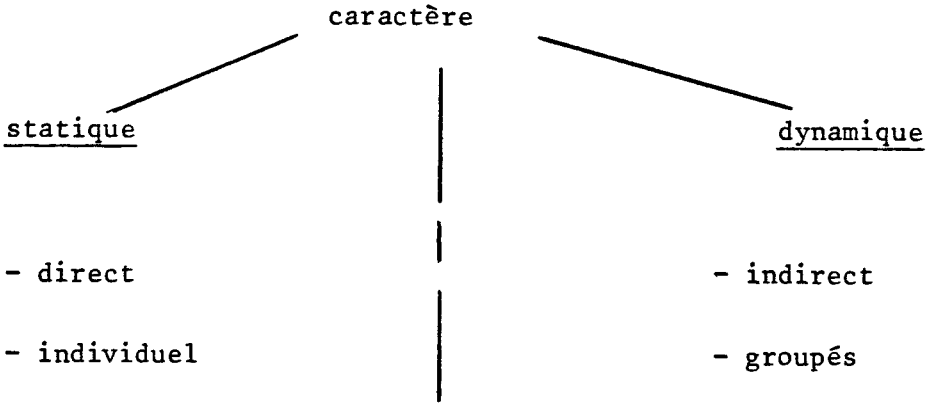
1.4.3.1. Attribut nature



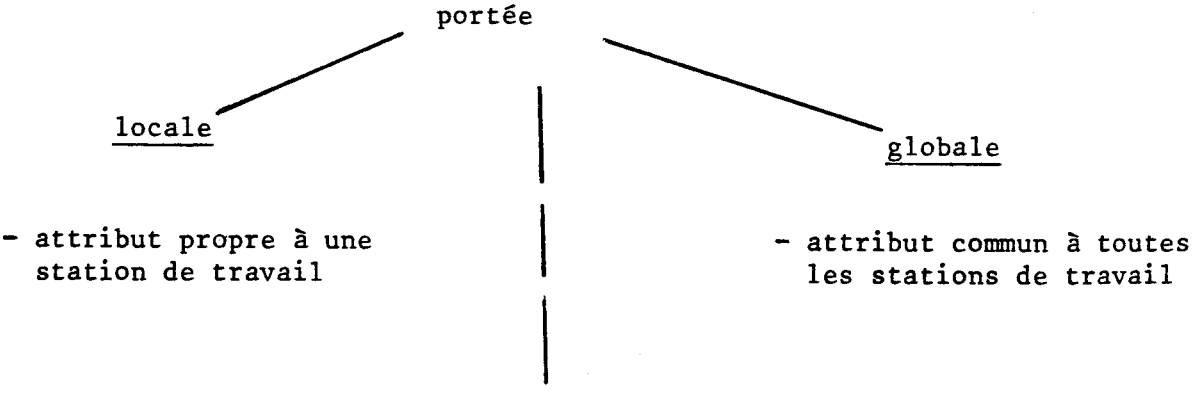
1.4.3.2. Attribut spécification



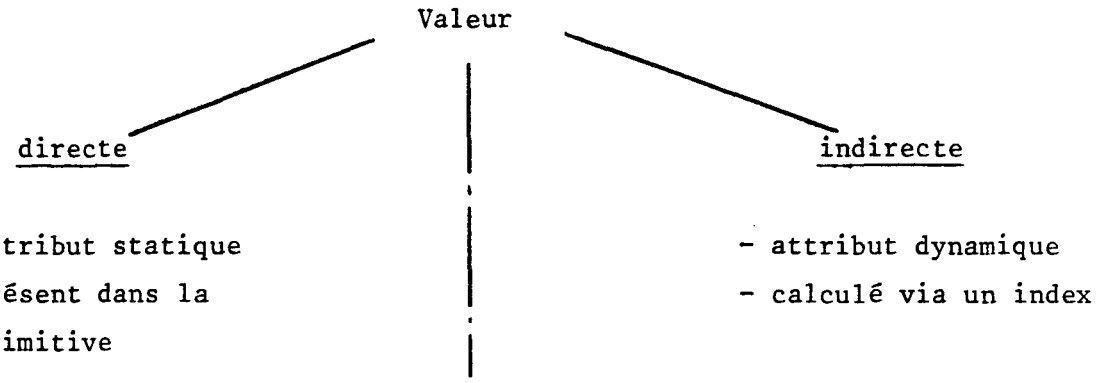
1.4.3.3. Attribut caractère



1.4.3.4. Attribut portée



1.4.3.5. Attribut valeur



1.4.4. Essai de classification [MER 79]

Divers systèmes de classification existent et dépendent des critères choisis pour cette classification. On peut citer comme critères, la liste non exhaustive suivante :

- type d'objets et images à produire (1D, 2D, 3D)
- nature des informations véhiculées (abstraites, réalistes, symboliques)
- techniques picturales utilisées (traits, points, tâches)
- origine et forme de description de l'image.

Pour notre part, nous avons choisi, dans un but de formalisation et d'essai de classification, le critère de description de l'image, ce qui nous a permis de distinguer deux grandes classes d'images fondées sur deux techniques :

i) la technique descriptive utilisée dans le domaine de l'analyse d'image

ii) la technique générative utilisée dans le domaine de la synthèse d'image

1.4.4.1. Technique descriptive

On regroupera sous cette appellation toutes les images obtenues à partir d'éléments extérieurs à un calculateur. A ce niveau (calculateur) on

se limitera à décrire les images en utilisant le plus souvent des matrices de points. Ce type d'image présente les caractéristiques suivantes :

- image traitée de façon séquentielle (utilisation de la matrice de description ligne/ligne) à cause de l'architecture des calculateurs traditionnels (architecture séquentielle).
- image plane et définie complètement à l'entrée du système graphique.
- nécessite des traitements globaux
- inadéquation pour décrire des formes géométriques

exemple :

photo satellite

1.4.4.2. Technique générative

Cette technique permet, à l'aide d'un calculateur, de produire (générer) des images.

On distinguera deux méthodes de génération

1.4.4.2.1. Méthode non-interactive

Elle correspond à la synthèse d'image de type algorithmique où la visualisation de résultats se fait après traitement.

Elle se caractérise par sa rigidité et par le fait que l'image est générée par programme sans intéraction avec l'utilisateur.

exemples : Résultats graphiques d'un calcul
gestion automatique de dessins

1.4.4.2.2. Méthode interactive

Comparativement à la méthode précédente et comme son nom l'indique, elle est la plus intéressante pour l'utilisateur. Elle correspond à la synthèse d'image de type interactive en ce sens que l'image est l'objet que l'on crée et que l'on manipule (transformations de différentes natures). Elle demande également des temps de réponse très courts aux actions de l'utilisateur.

Elle se caractérise par :

- la création progressive de l'image à l'aide d'outils (moyens d'entrée + procédures graphiques)

- la liaison entre les moyens d'entrée et la description des objets.

- la nécessité d'offrir un temps de réponse très court à l'échelle humaine.

- un traitement sélectif

- la présence de périphériques permettant une interaction

exemples : C A O
dessin animé

La notion d'image ayant été définie, nous allons étudier dans le paragraphe suivant, l'un des outils logiciels mis à la disposition des utilisateurs pour la création d'images, à savoir un langage graphique.

1.5. Les langages graphiques

1.5.1. Introduction

Un système graphique mettant en relation un calculateur avec un utilisateur, il est tout à fait normal de créer une interface entre les deux. Un des composants de cette interface se trouve être un langage mis à la disposition de l'utilisateur pour le développement et la manipulation du système. Le traitement graphique, d'une manière générale, requiert qu'un langage soit arithmétique et offre des possibilités graphiques, parce que les deux sont utilisées simultanément.

Partant de ce langage, créer une image consiste à choisir une correspondance entre un ensemble d'informations et les éléments du langage graphique [LUC 84]. Ce langage est défini par un vocabulaire (éléments de base du graphisme) et une grammaire (règles de composition). Ainsi cette création revient à définir un langage graphique, à analyser l'information à transcrire pour trouver les correspondances avec les éléments du langage [LUC 84]. On retrouve donc le processus classique de définition et d'utilisation d'un langage de programmation usuel (syntaxe + sémantique).

1.5.2. Types de langages [DUF 81, FOL 82, CAU 84]

L'étude de quelques langages graphiques nous a permis de les classifier en deux grandes familles.

i) les langages graphiques intégrés "LGI"

ii) les langages évolués graphiques "LEG"

1.5.2.1. Les langages graphiques intégrés

On désignera sous ce vocable les langages graphiques intégrés à un langage évolué hôte.

exemples : FORTRAN + CALCOMP
FORTRAN + GKS
PASCAL + GKS
C + GKS

Cette intégration se fait généralement par :

i) une adaptation des structures de données du langage hôte aux objets graphiques.

ii) des procédures d'interface entre le langage hôte et le langage graphique (appel de procédures graphiques, normes standards d'appel et de retour, normalisation des paramètres, etc...).

exemples :

FORTRAN + GKS

```
L10        SUBROUTINE DRFILE(file)
L20        REAL xarray(1000),yarray(1000)
L30        INTEGER file, length, i
L110  110  READ (file, 160,END=150)
      +    length, (xarray(i),yarray(i), i=1,length)
L120       CALL GPL (lenght, xarray, yarray)
L130       GOTO 110
L150  150  RETURN
L160  160  FORMAT (I10/(2F12,2))
L170       END
```

PASCAL + GKS

```
L10  PROCEDURE DRAW -FILE(VAR input-file:TEX
L20  VAR xarray, yarray : ARRAY[1..1000] of
     REAL ;
L30  i, length : INTEGER ;
L50  BEGIN
L60  RESET (input-file) ;
L70  WHILE NOT EOF  (input-file) DO
L80  BEGIN
L90  READ (input-file, length) ;
L100 IF (length > o ) THEN
L110 BEGIN
L120 FOR i : = 1TO  length DO READ (input-fil
                 xarray [i], yarray [i ]) ;
L130 POLYLINE (length, xarray, yarray) ;
L140 END ;
L150 END ;
L160 END {DRAW-FILE} ;
```

Cette famille de langages reste de loin la plus usitée car l'intégration du langage hôte au langage graphique est relativement facile. Elle présente néanmoins quelques défauts dont le plus important est celui de l'inadéquation des structures de données, pour certains langages hôtes, avec le type d'objet graphique, qu'est un objet à structure complexe [LOR 82].

Ainsi pour le langage FORTRAN intégré avec la norme GKS il existe toute une série de règles d'écriture qui viennent s'ajouter aux restrictions du langage FORTRAN et à ——— sa pauvreté en structures de données évoluées et dynamiques [SLA 84].

1.5.2.2. Les langages évolués graphiques

1.5.2.2.1. Présentation

Dans cette famille on parlera non plus d'intégration mais d'extension d'un langage évolué de programmation par des instructions graphiques.

L'exemple le plus typique est celui du langage PASCAL dont l'extension avec des structures de données et instructions graphiques a donné naissance aux langages MIRA [THA 81] et PASCAL/GRAF [BAR 81].

Exemples

- MIRA : Programme de création de cercles concentriques

```
PROGRAM cercle (INPUT, OUTPUT) ;
  TYPE
      concent = FIGURE (C : VECTOR ; R : REAL ; N : INTEGER) ;
                VAR
                   Cir : CIRCLE ;
                   pas : REAL ;
                    i  : INTEGER ;

                BEGIN
                   pas : = r/n ;
                   FOR i : = 1 TO n DO
                       BEGIN
                         CREATE Cir (c,r) ;
                         INCLUDE cir ;
                         r : = r - pas
                         END

                END ;
```

```
VAR
   Cercles : concent
BEGIN
   CREATE  Cercles (origin, 8.0, 10) ;
   DRAW    Cerles.
  END.
```

- PASCAL/GRAF : Ecriture du même programme en PASCAL/GRAF.

```
PROGRAM Cercle (INPUT, OUTPUT) ;
    VAR
       i : INTEGER ; plotter : OUTUNIT ;
       p : PICTURE ;
    BEGIN
       r : = 8.0 ; n :=10 ;
       p : = circle (r) ; pas : = r/n ;
       FOR i : = 1 TO 10 DO
           BEGIN
             e : = (r - pas)/r ;
             r : = r - pas
             p : = p OVER SCAL (e,p)
           END ;
       INIT (plotter, 0,10,0,10, TRUE) ;
       DRAW (plotter, p)
   END.
```

- SCAL (e,p) permet une mise à l'échelle de l'image p avec un rapport e
- OVER permet d'ajouter, à l'image p, l'image obtenue par SCAL (e,p).

1.5.2.2.2. Conclusion

Les avantages d'une telle méthode peuvent se résumer comme suit :

i) facilité d'extension

ii) universalité du langage

iii) portabilité plus facile qu'avec la méthode précédente

iv) richesse en concepts de base.

Les langages de programmation à étendre offrent l'avantage d'être relativement précis, puissants, flexibles et concis. Le principal inconvénient de la méthode d'extension, est le choix du langage. En effet, pour pouvoir garder toute la puissance et la facilité de la méthode, il faut disposer d'un langage relativement riche en structures de données et en instructions(exemple PASCAL).

1.5.2.3. Autres classifications

Un autre type de classification consisterait à classifier les langages selon les fonctions qui sont mises à la disposition des utilisateurs. Partant de ce point de vue, on aboutit à la classification suivante :

FONCTIONS	LANGAGES
vecteurs, caractères	CALCOMP, PLOT 10
Fichier graphique linéaire (segments)	G.K.S.
Fichier graphique structuré	PHIGS, GINO-F
élimination parties cachées, éclairage	MIRA

1.5.2.4. Conclusion

Bien que la syntaxe d'un langage uni-dimensionnel ne soit pas un moyen naturel pour exprimer un problème bi-dimensionnel, elle reste quand même

de loin, la seule technique utilisée. Néanmoins, des tentatives de recherches devraient être faites, à notre avis dans le sens des grammaires à deux niveaux ou orthogonales (type VAN WIIJNGAARDEN) pour déterminer si elles peuvent être d'un apport quelconque pour l'expression d'images à deux ou trois dimensions.

1.5.3. Caractéristiques d'un langage d'interface [DUF 81, FOL 82]

1.5.3.1. Point de vue de l'utilisateur

De ce point de vue, le langage de conversation doit être orienté vers l'utilisateur et non vers le calculateur. Il doit d'autre part établir une correspondance entre le vocabulaire utilisateur et les concepts de la machine.

Exemple :

> Un architecte est familier avec les notions de structure, plancher, poutres, murs, mais n'a aucune idée sur des notions telles que variables, listes, pointeurs, piles, etc...

Les premiers sont des concepts familiers à l'utilisateur, tandis que les seconds sont des notions qui lui sont étrangères et qu'il n'a ni le temps ni l'inclination à maîtriser.

1.5.3.2. Point de vue du concepteur

Comme pour tout langage, la conception d'un langage graphique doit, le mieux possible, répondre à un certain nombre de caractéristiques que nous résumons ci-dessous :

i) être conforme aux normes internationales (GKS en particulier)

ii) être efficace et complet :

- exprimer des commandes de manière précise (pas d'ambiguïté)
- permettre l'expression de toute idée de l'utilisateur
- établir une transparence du niveau machine
- avoir un nombre minimal de règles simples à apprendre
- permettre le traitement parallèle d'images
- être effectivement portable

iii) avoir une grammaire naturelle : éviter une grammaire complexe qui introduirait des discontinuités.

iv) être interactif

v) être apte à décrire, générer et manipuler des images.

vi) être flexible : pas de restrictions sur le nombre, le type et la dimension des images. Les éventuelles restrictions doivent être introduites dans une implémentation particulière et non dans le langage lui-même.

vii) être extensible : technique que l'on trouve dans les langages de programmation mais moins communément dans les systèmes graphiques.

viii) être universel : traitement d'une large classe d'images. Les éventuels traitements spécialisés peuvent être faits par la définition de sous-langages spécialisés.

1.5.4. Réalisation de logiciels graphiques [BAR 81, THA 81, CAU 84]

Nous aborderons cette réalisation sous l'aspect utilisation d'un langage de programmation, l'autre aspect (utilisation de bibliothèque de sous-programmes) présentant peu d'intérêt dans le cadre de notre étude.

Nous partirons de l'idée que nous avons un langage de programmation étendu pour traiter des problèmes graphiques. Cette extension se fait par l'adjonction syntaxique et sémantique :

i) d'objets graphiques

ii) d'opérateurs associés à ces objets

Cette adjonction peut se réaliser de deux manières :

i) écriture d'un compilateur ou interpréteur graphique (fig. 1.3)

ii) écriture d'un pré-compilateur permettant de traiter les objets graphiques et produisant un programme source résultant dans un langage connu (langage Cible) (Fig. 1.4).

1.5.4.1. Méthode du compilateur ou interpréteur

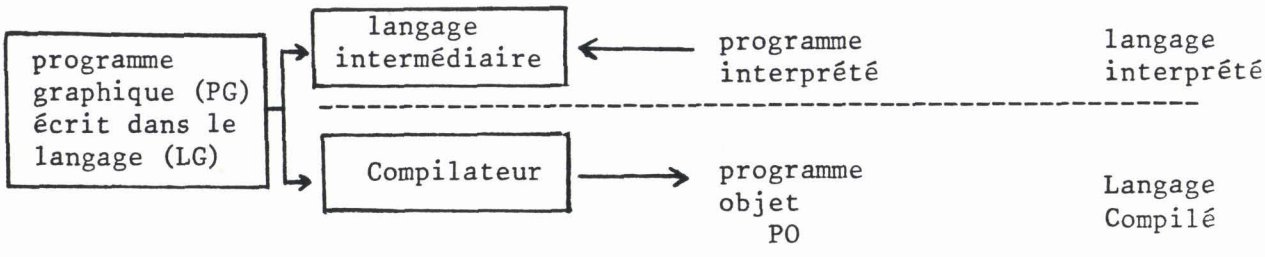


Fig. 1.3. Compilateur ou interpréteur

Avec cette méthode il ne s'agit pas de définir complètement un nouveau langage avec les contraintes d'apprentissage et de familiarisation, mais plutôt d'incorporer à un langage donné des outils graphiques. Comme exemples typiques de ces langages on peut citer les BASIC graphiques et le langage symbolique d'enseignement (LSE). Le lecteur intéressé trouvera dans [CAU 84] d'intéressants exemples sur ces langages.

1.5.4.2. Méthode du pré-compilateur

Nous rappelons que nous partons d'un langage renfermant un certain nombre d'objets classiques de programmation usuelle (entiers, réels, chaînes de caractères, structures de données, etc...). D'autre part nous disposons de mécanismes d'analyse de tels langages comme par exemple les compilateurs ou interpréteurs qui sont écrits suivant des techniques de compilation et d'interprétation bien précises. L'adjonction d'objets graphiques différents des objets classiques contenus dans le langage source risque de remettre en cause les techniques d'écriture des compilateurs ou des interpréteurs au point d'aboutir à leur réécriture complète.

La méthode du pré-compilateur permet de pallier à ce type d'inconvénients. Elle nécessite l'écriture d'un pré-compilateur qui aura l'avantage d'être portable sur tous les sites où le compilateur du langage SOURCE est implanté. Elle peut être schématisé comme suit (Fig. 1.4.)

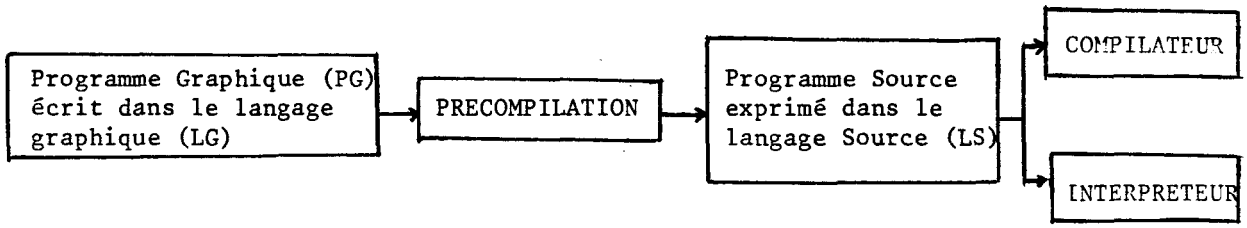


Fig. 1.4. Pré-Compilateur

Cette méthode est la plus usitée et a été choisie pour l'extension de certains langages tels que PASCAL [BAR 81, THA 81, YIP 84]. Ces deux méthodes bien que conceptuellement différentes permettent d'écrire des programmes d'application bien structurés, lisibles et dans un temps très court.

1.5.5. Formalisation

Indépendamment des travaux de normalisation des systèmes graphiques, on commence à s'intéresser de plus en plus à la formalisation des langages graphiques en utilisant les méthodes employées pour les langages de programmation usuels [CHU 78, PAR 82]. C'est une démarche qui nous semble originale pour un logiciel graphique. Il s'agit de savoir très précisemment "ce que signifie" chaque instruction sur les objets manipulés, c'est-à-dire sur l'environnement. Ceci permettra d'avoir, grâce à la possibilité de prouver chaque instruction, une référence formelle et admise de tous pour le langage considéré. Cette formalisation permettra d'améliorer la qualité du langage en fiabilité, extensibilité et transportabilité.

1.6. Les logiciels graphiques de base

1.6.1. Introduction

Nous terminerons ce premier chapitre par une présentation sommaire des standards graphiques qui représentent une partie importante de l'informatique graphique.

1.6.2. Classe de logiciels

On distingue généralement deux classes de logiciels :

i) les logiciels de base qui ont une certaine indépendance vis-à-vis de l'application et du matériel

ii) les logiciels d'application qui sont spécifiques à un domaine particulier et qui sont les plus répandus ;

1.6.3. Essai de normalisation

Les logiciels de base jouant un rôle très important dans l'organisation d'un système graphique, un grand nombre de travaux leur ont été consacrés et ont abouti à des normalisations [BON 82, FOL 82, END 84-a, MAC 84-a] que nous présentons ci-dessous.

1.6.3.1. G.S.P.C. (Graphic Standard Planning Committe)

Proposition américaine, plus connue sous le nom de CORE, c'est l'une des premières normalisations faite dans le domaine du graphique.

Elle a connue une refonte en 1979, avec l'utilisation des tubes à balayage télévision (Raster Scan). Elle s'articule autour des points suivants :

- manipulation d'éléments graphiques (traits, polygones)
- utilisation d'espace 2D et 3D
- utilisation de transformations géométriques
- possibilité de structuration des images.

1.6.3.2. G.K.S. (Graphical Kernel System) [END 84-a]

Proposition allemande, ce système représente les logiciels indépendants du matériel et configurables. Il constitue l'interface entre le programme d'application et le système graphique. Elle se caractérise par :

- une collection de programmes utilitaires appelables à partir d'un programme rédigé en FORTRAN ou en PASCAL (prochainement en ADA et en C)

- une utilisation de l'espace 2D uniquement
- une faiblesse dans la structuration
- une consultation restreinte d'objets

Il faut souligner ici que cette proposition a été acceptée en Juin 1984 comme norme internationale. Des travaux sont actuellement menés pour l'étendre à un espace 3D. Nous recommandons vivement au lecteur interessé de consulter l'article de ENDERLE [END 84-b] qui donne un panorama complet et détaillé des implémentations G.K.S. à travers le monde. Cette norme sert également de base pour le développement de nouvelles normes telles que PHIGS [HEW 84] qui est une proposition américaine et qui a pour objectif d'étendre la norme G.K.S. pour la représentation d'images 3D ainsi que pour la structuration des images.

1.6.4. Conclusion

Le principal effet de l'utilisation des langages graphiques standards dans des systèmes produisant des images a été de résoudre le problème d'indépendance du logiciel d'application vis-à-vis des machines utilisées. Ils ont le mérite de doter les programmes d'application d'un vocabulaire graphique standard [LAC 84]. On peut néanmoins noter les remarques suivantes :

- difficulté de normalisation
- diversité des approches

- mauvaise définition des concepts de base.

Ceci fait que la recherche dans ce domaine est loin d'être terminée, surtout dans la maîtrise des concepts de base des systèmes graphiques en général.

CHAPITRE 2 : STRUCTURES DE DONNEES ET SYSTEMES GRAPHIQUES

CHAPITRE 2

STRUCTURES DE DONNEES

ET

SYSTEMES GRAPHIQUES

****
****
****

2.1. Introduction

L'un des problèmes le plus important en programmation est celui de la structuration des données. Ce problème est particulièrement accentué en graphique car nous sommes en présence d'un domaine où le volume d'informations est assez important et où les notions de temps et d'espace mémoire prennent toute leur importance.

Ainsi la combinaison du graphique et des structures de données sera très avantageuse pour la construction de systèmes graphiques. Cette tentative de combinaison pose le problème suivant :

" Quelle structure de donnée doit-on choisir pour représenter et manipuler des structures graphiques tenant compte d'un certain nombre de contraintes ? ".

Diverses méthodes ont été utilisées et il est très difficile à priori d'avantager l'une par rapport à l'autre sans tenir compte de l'environnement dans lequel est choisie telle ou telle méthode, ainsi que des contraintes d'implémentation propres à cet environnement. Cette difficulté de choix provient essentiellement des systèmes graphiques eux-mêmes, d'où la difficulté d'adaptation des structures de données [WIL 74, WEL 80, LUC 84, EBO 85].

2.2. Problèmes posés par les systèmes graphiques

On peut résumer l'ensemble de ces problèmes en quatre classes :

2.2.1. Représentation

Les objets graphiques sont essentiellement des objets bi-dimensionnels (2D), et de plus en plus tri-dimensionnels (3D), alors que la mémoire des calculateurs est uni-dimensionnelle (1D). Cette caractéristique inhérente aux objets graphiques impose que les données de type 2D ou 3D soient transformées en une représentation 1D. En outre, il existe beaucoup de relations (liens) explicites dans un espace 2D qui doivent être préservées dans une représentation physique pour être conforme à l'image de base. Ce problème devient encore plus aigu lorsqu'il s'agit de représenter des espèces 3D en une représentation mémoire 1D et de les visualiser ensuite dans un espace 2D.

$$3D \xrightarrow[\text{Représentation}]{} 1D \xrightarrow[\text{visualisation}]{} 2D$$

On passe donc par différents espaces (3D, 1D, 2D) ainsi que par une série de transformations aussi différentes les unes des autres (représentation, affichage), sans oublier les propriétés propres à chaque espace, qui introduisent un décalage sémantique de l'objet image. Ceci pose nécessairement une certaine difficulté et complexité dans la réalisation d'un système graphique.

2.2.2. Calcul

Dans un espace à une dimension, le calcul se fait nécessairement en série. D'un point de vue calcul on a donc essentiellement un problème de conversion d'opérations 2D en une série d'étapes de traitement. Ces problèmes sont accentués par la nécessaire rapidité d'affichage sur un écran. D'autre part les techniques de calcul sont fortement marquées par les primitives de description.

2.2.3. Dynamicité des données

Dans un environnement graphique il existe beaucoup d'interactions très complexes, comme par exemple dans le cas de la Conception Assistée par Ordinateur (C.A.O.). Dans un tel environnement les structures de données changent constamment (niveau de dynamicité très élevé). Ceci pose un problème car il est très difficile de gérer des structures de données dynamiques en un temps relativement court, notamment avec les architectures séquentielles.

2.2.4. Présentation

Dans beaucoup d'applications graphiques, les utilisateurs ont besoin de travailler sur des niveaux de détail différents et ont besoin de faire varier également l'étendue de la présentation de la donnée à chaque niveau.

Exemples : * Cartes routières
- présentation des routes principales
- présentation des autres voies d'accès

* Imagerie médicale

Ainsi il est important d'être capable de définir des sous-ensembles de données et sélectivement de changer le détail associé avec chaque sous-ensemble. Réciproquement il doit être possible de combiner une donnée à partir de sources différentes en une entité affichable.

Tous ces problèmes de base sont liés aux difficultés de représentation des images dans un calculateur. Par conséquent, les structures de données (ou plutôt leur choix et leur adaptation) utilisées pour stocker ces images sont très importantes. La plupart des applications réalistes utilisent des bases de données très volumineuses et l'aspect graphique de la donnée constitue seulement une fraction des attributs de la donnée (représentation, liens, chemins d'accès, etc...) [WEL 80]. On peut donc affirmer que ces bases de données constituent beaucoup plus un support syntaxique qu'un support sémantique.

D'autre part il est important dans un système graphique d'avoir un maximum d'indépendance entre la donnée de base et le programme d'application pour que les éventuelles altérations ou réorganisations de ces données n'affectent pas le programme. Il est également nécessaire de permettre à l'utilisateur d'examiner différents sous-ensembles de la donnée, d'extraire, et d'intégrer des données sans difficulté.

2.2.5. Adaptabilité

Un certain nombre d'articles ayant été écrits sur les problèmes d'adaptabilité des structures de données au domaine graphique (Cf bibliographie relative au thème "structures de données et systèmes graphiques"). Nous nous contenterons d'en faire un bref rappel critique, dans le paragraphe suivant. Nous développerons un peu plus le modèle syntaxique et le modèle relationnel, auquel un intérêt particulier est attaché actuellement car il a des propriétés intéressantes [KUN 74, WEL 80, CHA 81, EBO 85] :

i) plus grande indépendance de la donnée, ce qui permet d'avoir différentes vues logiques de la donnée.

ii) organisation beaucoup plus proche de l'utilisateur que de la machine.

2.2.6. Importance des structures de données

Un grand nombre de structures de données peuvent être nécessaires au sein des processus de description et de visualisation. Dans un schéma classique de synthèse d'image on trouve [MAR 82] :

i) la maquette (2 ou 3D)
ii) l'épreuve constituant la vue (2D)
iii) la liste de visualisation

La multiplicité des structures de données dans de tels systèmes soulève quelques difficultés qu'on peut résumer en trois catégories :

i) la répartition ou la duplication des attributs de structures
ii) la mémorisation d'attributs à plusieurs niveaux, ce qui entraîne une surcharge de la mémoire
iii) la complication de mise en oeuvre de synthétiseur par la présence de divers mécanismes de gestion des structures de données. C'est ainsi que la puissance et la souplesse d'utilisation d'un système graphique en général dépendent fortement de la structure de donnée utilisée et de son organisation, d'où l'importance particulière qui leur est accordée. LUCAS dans [LUC 84] note d'ailleurs avec une certaine curiosité que l'étude fine des algorithmes et des structures de données se découvre une grande importance. Il y a là un paradoxe dans la mesure où la présentation d'images complexes, avec un haut degré de réalisme, pourrait faire croire que l'on sait tout très bien faire. Il n'en est rien, et les travaux sur la complexité des algorithmes graphiques et la complexité géométrique commencent seulement à prendre leur essor.

2.3. Modèles de description de scènes

Nous présenterons dans ce paragraphe deux modèles permettant quelques formalismes avec lesquels les descriptions peuvent être générées et exprimées. Ces formalismes ne sont pas spécialisés pour une classe particulière de scènes.

Ces modèles sont basés l'un sur la théorie des langages [SHA 79], l'autre sur les concepts de bases de données [OMO 79, WEL 80, CHA 81, FOI 84, SPO 84, EBO 85, RIE 85].

Le choix de tels modèles répond aux considérations suivantes :

i) le premier modèle s'intègre dans l'orientation que nous avons donné à nos travaux, à savoir l'étude des systèmes graphiques sous l'aspect langage. Nous avons d'ailleurs proposé dans cette thèse, une investigation

pour une éventuelle utilisation des grammaires à deux niveaux (grammaires orthogonales [JOU 84-a]) pour l'expression de langages graphiques.

ii) le deuxième, à cause de l'intérêt sans cesse grandissant accordé aux bases de données, notamment celles basées sur le système relationnel, pour leur utilisation en tant que structure de données dans les systèmes graphiques.

2.3.1. Modèle syntaxique [SHA 79]

Pour illustrer ce modèle on partira d'un exemple simple d'une scène composée d'une façade de maison avec un arrière-plan composé d'un cercle représentant le soleil (Fig. 2.1.).

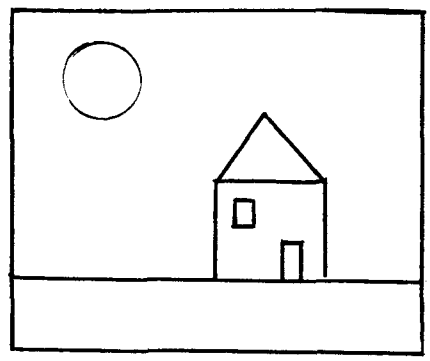


Fig. 2.1. Exemple de scène.

Avec un niveau de description très grossier, la scène peut être caractérisée simplement comme : " façade maison et soleil".

Puisque nous sommes intéressés par la structure de la scène,nous pouvons élaborer un peu plus cette structure et décrire la scène comme "la façade est à droite du soleil". Procédant par description fine, la façade peut être décrite comme une collection de trois quadrilatères et d'un triangle. On peut donc pousser cette description jusqu'au niveau de détail désiré (exemple : collection de segments).

Ce processus de raffinement successif de la description structurelle d'une scène marque une ressemblance avec le processus utilisé pour obtenir une description structurelle d'une phrase d'un langage (le mot langage étant pris au sens large).

On peut donc considérer la notion de grammaire (au sens de la théorie des langages) comme un formalisme avec lequel peuvent être décrites des images.

Ainsi l'approche grammaticale présente un attrait pour mettre au point une théorie potentiellement pratique dans les problèmes de description et d'analyse de scène.

La principale difficulté conceptuelle rencontrée dans l'extension d'une syntaxe formelle aux espaces bidimensionnels, provient d'un fait fondamental : une ligne uni-dimensionnelle a un ordre naturel tandis que les plans à deux dimensions n'en disposent pas. Par exemple, l'opération naturelle avec les chaînes uni-dimensionnelles de symboles est la concaténation. Il n'existe pas d'analogie naturelle en deux dimensions. Pour être plus explicite en graphique, considérons de nouveau la scène décrite plus haut :

Un arbre d'analyse de cette scène peut être défini comme suit : (Fig. 2.2.)

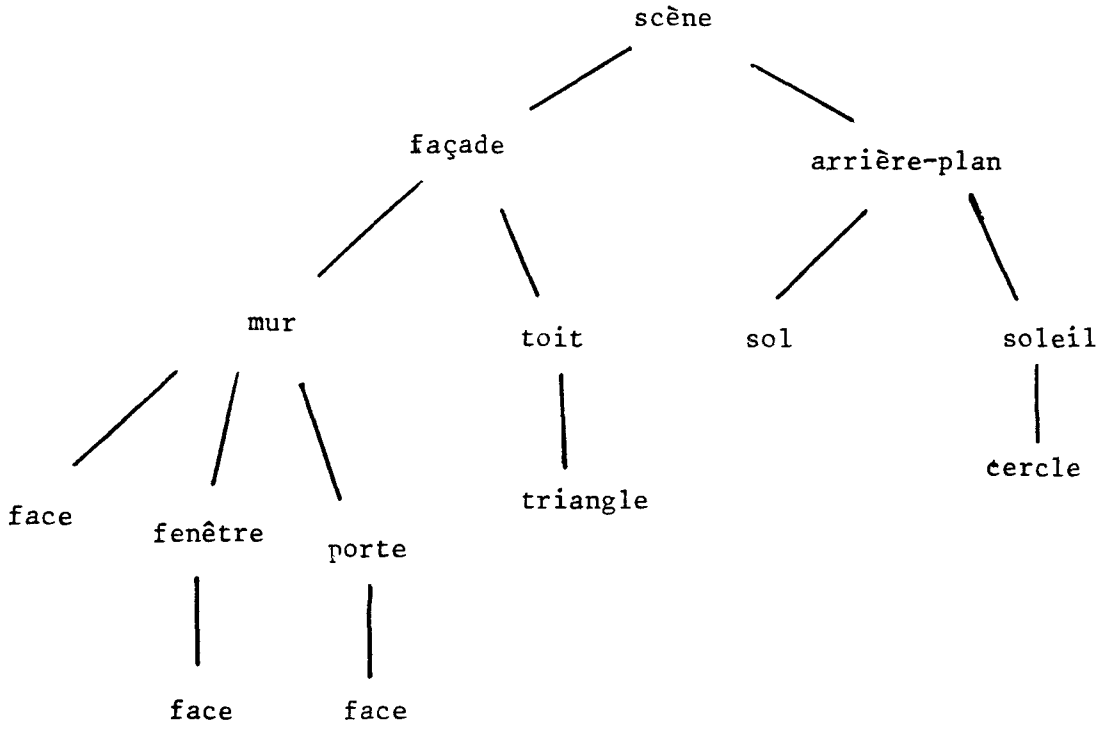


Fig. 2.2. Arbre d'analyse

Cependant même si nous voulons spécifier de manière précise les descriptions de quelques primitives comme "face", l'arbre décrit la scène vaguement. Par exemple, les trois faces peuvent être décrites ensembles par plusieurs chemins, mais seules quelques une d'entre elles représentent "fenêtre", "porte", "mur".

Ce problème posé par l'absence d'un ordre naturel pour le plan a été approché par un certain nombre de méthodes :

La plus simple consisterait à considérer un ensemble uni-dimensionnel de points muni de la fonction "ordre naturel". Comme exemple trivial de cette approche on peut définir un quadrilatère comme :

quad ::= segment ⊕ segment ⊕ segment ⊕ segment.

où ⊕ signifie la concaténation (il est sous-entendu que la concaténation doit être fermée sur elle-même).

Cette approche est très limitative dans le processus de description. Elle implique pour éviter ces limitations d'ajouter un certain nombre de fonctions de relations.

Les méthodes syntaxiques soulèvent quelques problèmes, dont quelques unes sont mentionnées ici :

i) difficulté de détection des primitives dans une scène
ii) inadéquation pour des classes variées d'images.

L'avantage fondamental des modèles syntaxiques est le principe de récursivité dans la description. Par exemple dans le cas d'analyse de scène, une grammaire est plus appropriée lorsque la scène est construite à partir d'un ensemble de primitives par application récursive d'un petit ensemble de règles de réécriture. Quand ceci n'est pas possible, c'est-à-dire quand la puissance de la récursivité ne peut être appliquée, alors la grammaire devient simplement une méthode formelle pour décrire exhaustivement à l'aide de symboles (mots du langage) le contenu entier d'une image, et l'avantage de la compacité de la description est perdu.

2.3.2. Modèle bases de données

Trois modèles classiques sont utilisées pour la réprésentation de données alphanumériques :

i) le modèle hiérarchique
ii) le modèle réseau
iii) le modèle relationnel

Notre travail n'ayant pas pour but une étude complète de ces modèles, nous nous contenterons d'en énoncer les principes fondamentaux ainsi que leurs apports éventuels en graphique.

2.3.2.1. Modèle hiérarchique [WIL 71, CHA 81, DEL 82, FOI 84, LUC 84, MIR 84, EBO 85]

Ce modèle utilise une structuration classique fondée sur la notion d'arbre qui permet de définir une hiérarchie de dépendance entre les divers éléments de la scène (fig. 2.3.).

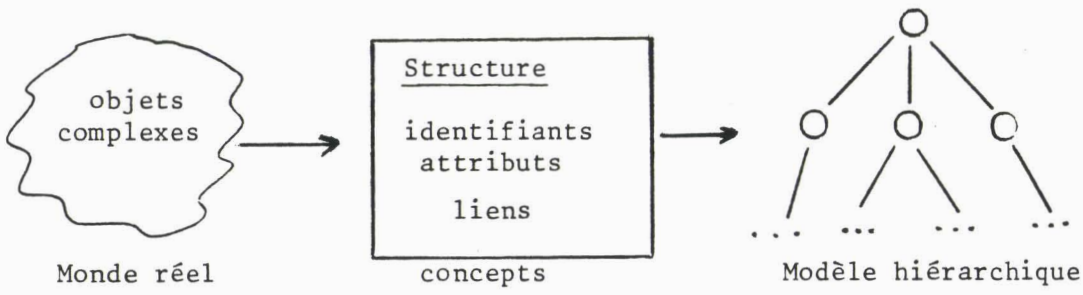


Fig. 2.3. Du monde réel au modèle hiérarchique

Ce type de modèle permet un stockage de l'image par décomposition. Différentestentatives ont été faites avec un tel modèle pour la représentation d'objets graphiques. A titre d'exemple on citera les travaux de STIES, SANYAL et LEIST [STI 76] qui décrivent un système hiérarchisé d'images où un objet est décrit en terme de figure, description symbolique et relations entre objets. KLINGER et al [OMO 79] ont proposé une structure de donnée hiérarchique pour stocker des images par décomposition régulière sous forme de quadruplets adjacents.

2.3.2.1.1. Avantages et inconvénients

Ces avantages et inconvénients sont vus notamment sous l'aspect utilisation du modèle en graphique.

a/ avantages

i) adéquation du modèle à une structure arborescente

ii) simplicité du modèle, ce qui permet une implémentation relativement facile

iii) modèle utilisé par un nombre élevé de systèmes de gestion de bases de données commercialisables sur le marché.

b/ inconvénients

i) impossibilité de représenter des liens maillés (N:m) inter-entités. Cette représentation n'est possible que par la duplication des données physiques ou par la définition de pointeurs logiques. Cette technique ne peut être utilisée en graphique, car on met en jeu un nombre élevé d'objets complexes (base volumineuse, temps de traitement élevés).

ii) opérations d'insertion et de destruction très complexes dûes à l'arborescence.

iii) indépendance logique très réduite (forte connection avec l'application) ; ce qui est en contradiction avec le principe d'indépendance des systèmes graphiques

iv) limite en conception et en manipulation d'objets graphiques complexes.

On remarque donc que ce type de modèle s'adapte très mal à des systèmes graphiques qui mettent en jeu un nombre assez élevé d'objets complexes demandant parfois un niveau de détail très fin avec une contrainte en ce qui concerne le temps de réponse.

2.3.2.2. Modèle réseau [DEL 82, FOI 84, MIR 84, EBO 85]

Ce modèle, qu'on peut considérer comme une généralisation du modèle hiérarchique, permet de représenter une structure sous forme d'un réseau d'où le nom du modèle) connectant les entités entre elles à l'aide de pointeurs logiques (Fig. 2.4.).

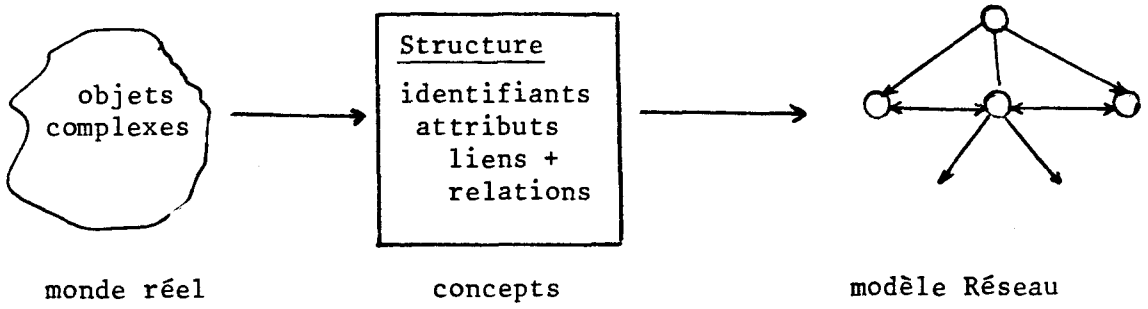


Fig. 2.4. Du monde réel au modèle réseau.

Les concepts de base de ce modèle sont :

- la notion d'enregistrement
- la notion de liens

2.3.2.2.1. Avantages et inconvénients

a) avantages :

i) accroissement de la flexibilité du système grâce à sa structure symétrique

ii) représentation naturelle de liens maillés

b) inconvénients

i) structure figée par l'absence d'une indépendance vis-à-vis des stratégies d'accès : les structures sont spécifiées en termes d'accès aux données.

ii) procéduralité importante des langages de manipulation. L'utilisateur doit "naviguer" dans le réseau logique consituté par les enregistrements et les chaînes de pointeurs.

iii) limité en conception d'objets graphiques complexes, comme pour le modèle hiérarchique

2.3.2.3. Modèle relationnel [KUN 74, WEL 80, CHA 81, DEL 82, FOI 84, MIR 84, EBO 85]

En partant de l'idée qu'une image est une forme de réprésentation d'une donnée quelconque du monde réel, une structure de donnée graphique peut être vue comme un modèle mathématique de cette donnée. Les concepts d'un tel modèle permettent de décrire les propriétés de cette donnée (Fig 2.7.).

Indépendamment des propriétés de description des éléments individuels composant cette donnée, il décrit une propriété importante, à savoir les relations existantes dans la donnée (Fig. 2.5.)

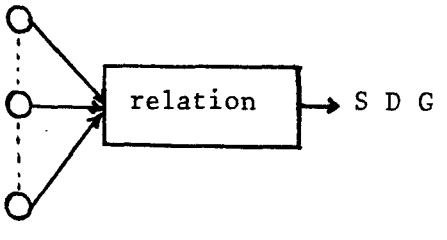


Fig. 2.5. Définition schématique d'une structure de donnée graphique (S D G)

Ainsi si une relation du monde réel peut être modelée par une structure de donnée graphique, chaque exemplaire d'une entité peut l'être également. Par exemple la relation entre un cercle, son rayon et son centre est une relation fixe, parce que chaque cercle a un centre et un rayon. Par conséquent une telle relation peut être modelée par une structure de donnée (Fig. 2.6.).

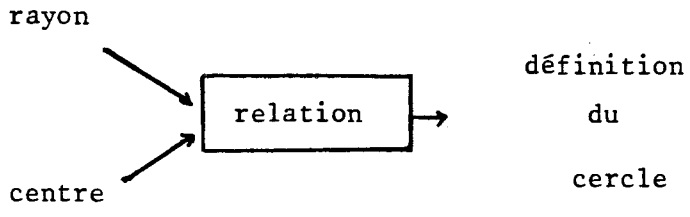


Fig. 2.6. Définition d'un cercle

A l'inverse des relations dites fixes, il existe des relations temporaires du monde réel, fonctions des notions d'espace et de temps, qui ne peuvent être modelées.

La particularité d'un tel modèle est qu'une structure étant très complexe, ses fonctions de représentation peuvent être rendues simples.

Ces modèles relationnels rencontrent un si grand succès qu'un certain nombre de travaux ont été consacrés à leur adaptation au graphique.

Les concepts de ce modèle peuvent se résumer comme suit :

i) une description simple, sous forme tabulaire des entités

ii) une manipulation non procédurale des données

iii) une représentation des données basée sur la notion mathématique de relation

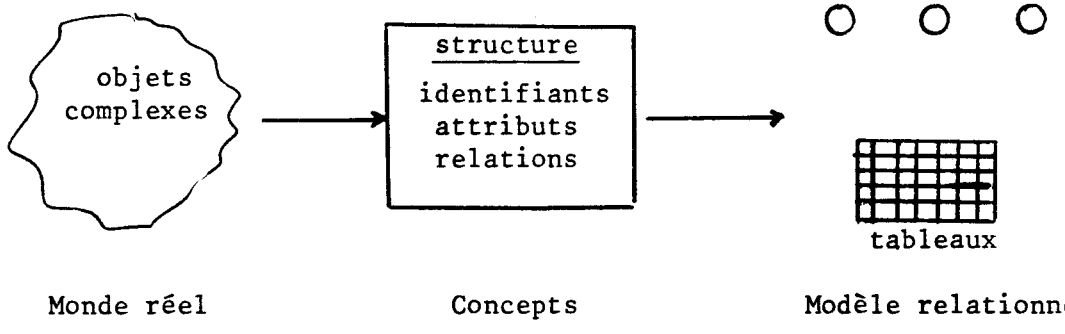


Fig. 2.7. Modèle relationnel

Comme travaux consacrés aux bases de données on peut citer ceux de [KUN 74] qui a proposé une approche relationnelle d'un système de gestion de bases de données pour décrire un ensemble d'images complexes avec texture et couleur.

WELLER et WILLIAMS [WEL 80] ont développé un système appelé PICTURE BUILDING SYSTEM, qui a intégré une base de donnée à un système graphique.

BOEING a étudié un prototype d'interface permettant d'afficher des objets géométriques stockés dans une base de donnée relationnelle [SPO 84].

En ce qui concerne les aspects modélisations, on peut citer les travaux de l'Université de Technologie d'Helsinski [MAN 82], de FOISSEAU et CHOLVY [FOI 84] de RIEU-RIALHE [RIE 85] et de WOO [WOO 84].

Ces différents travaux proposent des modèles qui permettent de faciliter les taches du concepteur (création, traitements), et d'augmenter la créativité (aptitude du modèle à la déduction).

2.3.2.3.1. Du modèle hiérarchique ou réseau au modèle relationnel

Certaines tentatives de combinaison du modèle relationnel avec les autres modèles ont été faites. Parmi celles-ci on peut citer celle de [CHA 79] qui a proposé une combinaison du modèle relationnel avec le modèle hiérarchique . Dans cette proposition il insiste sur le fait qu'il faut distinguer l'image logique (niveau utilisateur) de l'image physique (niveau machine).

On peut souligner que la migration des bases hiérarchiques et réseaux vers des modèles relationnels est toujours possible. Elle pose néanmoins d'énormes difficultés si des incohérences existent dans les modèles de base (hiérarchique ou réseau).

2.3.2.3.2. Avantages et inconvénients

a) Avantages

i) modèle satisfaisant
ii) facilité d'utilisation (mise en oeuvre non-procédurale)
iii) indépendance vis-à-vis des programmes
iv) simplicité de description
v) puissance de structuration et de représentation
vi) adapté à des interfaces non-procéduraux.

b) Inconvénients

i) demande une adaptation aux objets graphiques (notion d'objet complexe [LOR 82])

ii) ne permet pas de prendre en compte telle quelle la sémantique de certains attributs

iii) peu de systèmes de gestion de bases de données relationnelles offrent une solution satisfaisante au problème de l'intégrité et plus particulièrement à l'intégrité référentielle (cohérence d'une donnée avec les autres informations de la base).

iv) existence de certains problèmes sémantiques : par exemple l'ajout d'un lien arborescent intra-relation entraînera une modification sensible du schéma (éclatement de la relation initiale)

v) exige l'existence d'un langage relationnel c'est-à-dire non-procédural

vi) existence d'un nombre réduit de systèmes de gestion de bases de données relationnelles sur le marché.

2.4. Relations entre structures de données et systèmes graphiques [RÖT 80, MAR 82, CAU 84]

2.4.1. Introduction

Le paragraphe précédent ayant fait le point en ce qui concerne les structures de données, nous nous proposons, dans celui-ci, de mettre en évidence les relations existantes entre ces structures de données et les différents composants d'un système graphique.

Dans un système graphique interactif, l'affichage d'une image n'est qu'une représentation partielle d'objets beaucoup plus complexes, qui sont mémorisés dans une structure de données [RÖT 80] (fig. 2.8.).

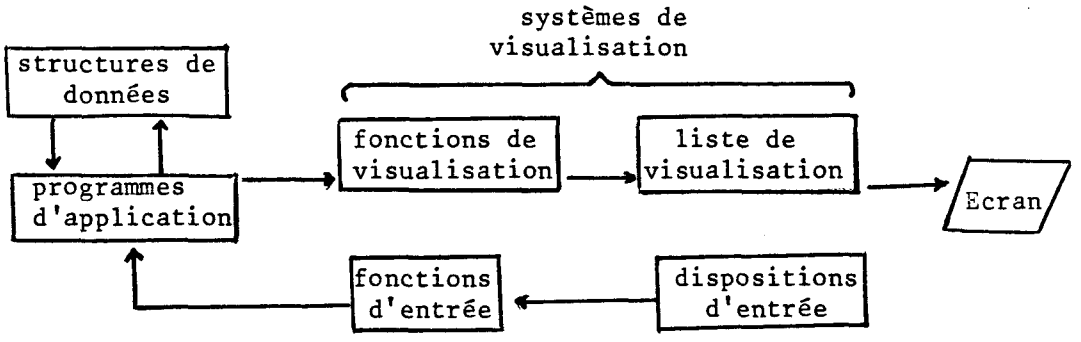


Fig 2.8. Schéma général d'un système interactif.

Dans un souci de cohérence, nous essayerons de mettre en évidence les relations entre structures de données et systèmes graphiques en partant de la notion d'écran. Ainsi ces relations seront étudiées pour les écrans à balayage cavalier, ainsi que pour les écrans utilisant un balayage de trame.

2.4.2. Ecrans à balayage cavalier

Il faut rappeler que pour ce type d'écrans, l'image peut être mémorisée soit directement sur l'écran (tube mémoire type TEKTRONIX) soit dans une mémoire intermédiaire (mémoire d'entretien) dans le cas d'utilisation d'une liste de visualisation.

Dans un tel cas, on peut aboutir à différents modèles de systèmes de visualisation suivant l'emplacement de cette mémoire d'entretien dans la suite des étapes de visualisation.

Les paragraphes suivants se proposent de faire un inventaire critique des modèles utilisés actuellement dans les systèmes graphiques interactifs.

2.4.2.1. Modèle sans mémoire intermédiaire

Dans ce modèle on utilise directement la structure de donnée de l'application pour afficher l'image (fig. 2.9.). On rappelera pour les besoins de l'histoire que cette solution a été adoptée en 1968 sur le LDS-1 de EVANS et SUTHERLAND [RÖT 80].

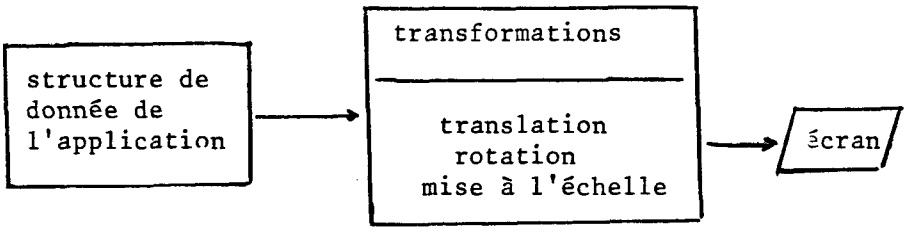


Fig. 2.9. Modèle sans mémoire intermédiaire

Les caractéristiques d'un tel modèle peuvent se résumer comme suit :

i) utilisation directe de la structure de donnée pour afficher l'image

ii) correspondance totale entre image mémorisée et image affichée

iii) limitation de la structure de donnée (taille) dûe au format exigé pour l'affichage

iv) réalisation matérielle des fonctions de visualisation à cause de la contrainte de rafraîchissement

2.4.2.2. Modèle avec fichier intermédiaire structuré

Dans ce modèle, le fichier intermédiaire est obtenu à partir de la structure et ne contient qu'une description de l'image affichée (fig. 2.10.)

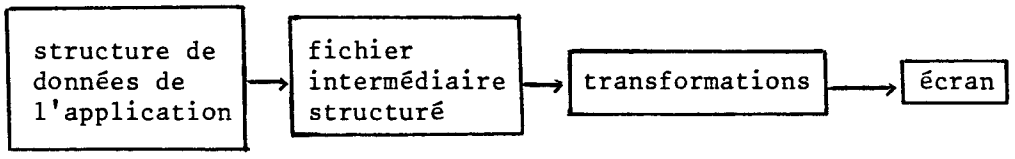


Fig. 2.10 Modèle avec fichier intermédiaire structuré

Les primitives d'un tel fichier sont limitées aux seuls vecteurs. En fait ce fichier est un fichier structuré car il renferme une description hiérarchique de l'image.

Les caractéristiques d'un tel modèle se résument comme suit :

i) description hiérarchisée de l'image

ii) raffraichissement de l'image à partir du fichier intermédiaire

iii) taille et format des structures de données de l'application non limitées

iv) différence entre image mémorisée et image affichée

v) utilisation dans les systèmes vectoriels récents

vi) mise à jour obligatoire de deux structures de données pour toute opération de modification de l'image (structure de donnée + fichier intermédiaire).

2.4.2.3. Modèle avec fichier intermédiaire linéaire

A l'inverse du modèle précédent, on utilise ici un fichier intermédiaire qui ne contient plus une description hiérarchique de l'image. Le modèle est utilisable pour des systèmes moins performants (fig. 2.11.).

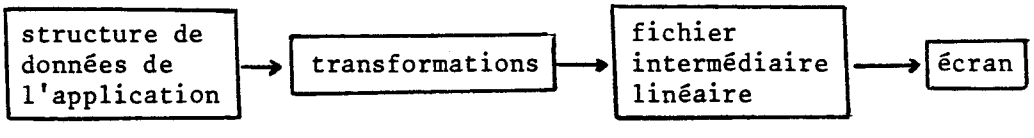


Fig. 2.11. Modèle avec fichier intermédiaire linéaire

Dans ce modèle les fonctions de visualisation sont réalisées au moyen de logiciels, ce qui a pour effet de rendre impossible le rafraîchissement de l'image en temps réel. Un fichier intermédiaire après transformations s'impose. Par analogie au fichier structuré du modèle précédent, c'est un fichier linéaire dont les caractéristiques se résument comme suit :

i) contient une liste de vecteurs regroupés éventuellement par segments affichés continuellement
ii) organisation linéaire des vecteurs ou des segments
iii) facilité de mise à jour du fichier (insertion, suppression, etc...).

Ce modèle, par son aspect économique, reste à l'heure actuelle le plus répandu. Il est ainsi utilisé par la norme américaine CORE [SIG 79].

Un modèle dérivé consiste à utiliser seulement le fichier intermédiaire sans structure de données. Partant de cette idée, SPROULL propose un modèle basé sur des procédures graphiques [NEW 79]. Dans un tel modèle, une image n'est plus décrite par sa structure hiérarchique (au sens de sa structure de donnée) mais plutôt par des procédures graphiques représentant les primitives graphiques usuelles. L'imbrication de ces procédures décrit donc la hiérarchie de l'image. Par contre toute modification de l'image implique une recompilation et une exécution de toutes les primitives de description en vue d'un affichage.

Nous avons choisi de développer une telle méthode car comme on le verra dans la troisième et quatrième partie de ce travail, le langage pour lequel nous avons opté est un langage basé sur la notion de fonction (LISP).

Ainsi la description d'une image dans un tel langage sera exprimée par une suite de fonctions dont la composition décrit la structure hiérarchique de l'image. On peut donc souligner, d'ores et déjà, que la structure du langage choisi a été l'un des arguments de choix d'une telle méthode. On notera cependant que cette approche est possible avec tous les langages évolués renfermant le concept de procédure ou de fonction.

2.4.3. Ecrans à balayage de trame

Comparativement au type d'écrans précédent, il existe peu de modèles pour les écrans à balayage de trame du fait qu'ils ont été introduits

dans les systèmes graphiques bien après les écrans à balayage cavalier. A la différence de ces derniers, les écrans à balayage de trame utilisent un niveau supplémentaire de mémorisation de l'image caractérisé par la mémoire de trame. Ainsi l'image est mémorisée à deux niveaux :

i) dans la structure de donnée
ii) dans la mémoire de rafraîchissement

Bien que ces deux niveaux de mémorisation soient largement suffisants dans un système graphique, on peut envisager pour certains types d'applications, un niveau supplémentaire de mémorisation, afin de rester homogène avec les systèmes vectoriels.

2.4.3.1. Modèle sans mémoire intermédiaire

C'est le modèle le plus simple qui peut, pour des applications de bas niveau, se limiter à un modèle qualifié de minimal et comprenant une mémoire de rafraîchissement et un écran (Fig. 2.12).

Fig. 2.12. Modèle sans mémoire intermédiaire

Il faut noter que pour la reconstitution d'une image affichée sur un écran à balayage de trame, la structure de donnée s'avère indispensable (difficulté de régénération).

2.4.3.2. Modèle avec fichier intermédiaire structuré

L'intérêt principal d'un tel modèle est son degré de performance dans le cas d'un système interactif (Fig. 2.13)

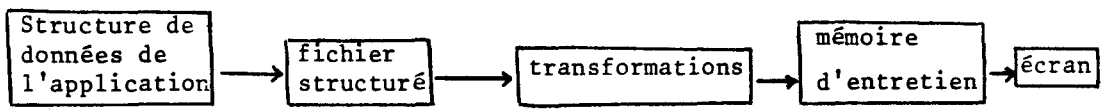


Fig. 2.13. Modèle avec fichier structuré

Dans ce modèle, les outils de traitement en pipe-line ou en parallèle trouvent toute leur puissance, ce qui a pour effet de mettre à jour la mémoire d'entretien à partir du fichier structuré. D'autre part pour des applications qui ne traitent que des objets visibles sur écran. Cette mémorisation à trois niveaux (structure de données, fichier structuré et mémoire de rafraîchi-sement) peut être évitée.

2.4.3.3. Modèle avec fichier intermédiaire linéaire

Dans ce modèle, le fichier linéaire joue le même rôle qu'un fichier structuré (Fig. 2.14.)

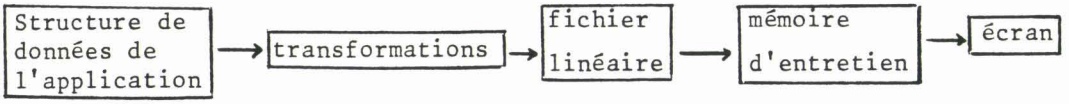


Fig. 2.14. Modèle avec fichier linéaire

Le fichier linéaire permet une mémorisation de l'image, une reconstitution assez facile de l'image affichée ainsi qu'une mise à jour rapide. Par contre la modification d'un objet de l'image nécessite la reconstitution de l'ensemble du fichier linéaire.

2.4.4. Conclusion

En guise de conclusion, nous terminerons ce chapitre par un tableau de synthèse sur les différents modèles de mémoire intermédiaire utilisables dans les systèmes graphiques utilisant la technique du balayage cavalier ou la technique du balayage de trame.

BC ⟶ Balayage cavalier
BT ⟶ Balayage de trame
SD ⟶ Structure de donnée

propriétés et fonctionnalités / type de balayage	taille image affichée / image mémorisée		nombre de mise à jour SD ⟶1 SD+fichier⟶2		fonctions de visualisation		taille SD/image affichée		niveau affichage		type de traitement	
	BC	BT	BC	BT	BC	BT	BC	BT	BC	BT	BC	BT
modèle sans fichier intermédiaire	1	1	1	1	cablées	cablées	limitée	pas de limitation	SD	mémoire d'entretien	séquentiel	séquentiel
modèle avec fichier structuré	< 1	< 1	2	2	logicielles ou mixtes	mixtes	pas de limitation	pas de limitation	fichier intermédiaire	mémoire d'entretien	-	pipe-line parallèle
modèle avec fichier linéaire	1	1	2	2	logicielles	mixtes	pas de limitation	pas de limitation	fichier intermédiaire	fichier intermédiaire	séquentiel	séquentiel

TABLEAU COMPARATIF DES MODELES DE MEMOIRE INTERMEDIAIRE

CHAPITRE 3 : LES LANGAGES DE PROGRAMMATION

CHAPITRE 3

LES LANGAGES DE PROGRAMMATION

****
****
***

3.1. Introduction

Après avoir passé en revue, dans le chapitre précédent, les problèmes de structures de données ainsi que les solutions apportées dans les systèmes graphiques, nous nous intéresserons, dans ce chapitre, beaucoup plus à l'aspect langages de programmation qu'à l'aspect graphique. Notre expérimentation étant réalisée avec un langage conceptuellement différent des langages de programmation usuels, il nous a semblé opportun de faire une analyse critique des langages de programmation actuels, qui utilisent une approche de programmation procédurale ou impérative. Nous présenterons par la suite, les langages de programmation non-procéduraux, appelés également langages de la 5ème génération, dont fait partie le langage que nous avons choisi pour notre expérimentation. (LISP).

3.2. Les langages de programmation en graphique

3.2.1. Introduction

Nous présenterons dans ce paragraphe, un inventaire des langages de programmation les plus usités en graphique. Ces langages font partie de la classe des langages procéduraux ou impératifs dont nous verrons les inconvénients dans les paragraphes suivants.

3.2.2. Les langages de programmation graphiques

La majeure partie des langages de programmation utilisables en graphique représentent soit une intégration, soit une extension des langages de programmation usuels (Cf. 1.5.).

Le tableau suivant donne une liste de ce type de langage.

Nom du langage	langage Source	Année	Auteurs
BASIC PLOT 10	BASIC		langages de bas niveau développés par des constructeurs. (ex. TEKTRONIX)
BASIC PLOT 50	BASIC		
FORTRAN 3D	FORTRAN	≃ 1976	I.N.R.I.A.
GRAF	FORTRAN	1967	A. HURWITZ, J.P. CITRON J.B. YEATON
CPL/1	PL/1	1976	G. F. PFISTER
GPL/1	PL/1	1971	D.N. SMITH
MIRA - 3D	PASCAL	1981	M.N, D. THALMANN
MIRA - SHADING	PASCAL	1984	M.N, D. THALMANN
PASCAL/GRAF	PASCAL	1981	W. BARTH
GRAPHEX 68	ALGOL 68		
ALGRAPH 68	ALGOL 68		
ESP	SNOBOL	1975	L.G. SHAPIRO
	ALGOL-60	1976	B. JONES
APL-G	APL	1972	W.K. GILOI, J. ENCARNACAO W. K ESTNER
LOGO-G	LOGO	1973	W.M. NEWMAN
PICTURE-BALM	LISP	1980	G.B. GOATES, M.L. GRISS G.J. HERRON

Langages de programmation graphiques

A la lecture de ce tableau on remarque, qu'à deux exceptions près, l'ensemble des langages graphiques sont développés à partir de langages procéduraux utilisables en programmation classique.

D'autre part, les standards graphiques, tels que G.S.P.C. CORE, G.K.S sont appliqués seulement avec la classe des langages impératifs avec une intégration d'un langage hôte pour le graphique (FORTRAN, PASCAL, etc...).

Ces standards graphiques ont permis une indépendance des logiciels envers le matériel graphique et une approche de haut niveau autorisant un degré croissant de portabilité. Néanmoins, ceci est insuffisant car les langages procéduraux présentent un certain nombre d'inconvénients et de contraintes qui dérivent notamment des architectures sur lesquelles ils sont bâtis, à savoir les architectures VON-NEUMANN.

3.3. Les nouvelles approches en programmation

3.3.1. Introduction

Avant d'aborder la présentation de la nouvelle approche en programmation, nous mettrons en évidence les inconvénients de l'approche classique utilisée en programmation, à savoir l'approche algorithmique et déterministe. Ces inconvénients sont en partie dûs au modèle de VON-NEUMANN sur lequel sont implémentés les langages de programmation usuels [WEG 76, SLI 77, BAC 78, SLI 84, SLI 85].

3.3.2. Approche algorithmique

Ce type d'approche permet la résolution de problèmes du monde réel décrits sous une forme mathématique (algorithmique) et dans un référentiel déterministe. Cette approche a permis le développement de langages de programmation sur la base des caractéristiques architecturales du modèle de VON-NEUMANN qui est un modèle contraignant pour la résolution de certains types de problèmes.

En effet, ces langages de programmation ont une sémantique procédurale, c'est-à-dire que la résolution d'un problème contraint le programmeur à spécifier explicitement les taches que la machine doit accomplir l'une après l'autre, avant d'aboutir à la résolution du problème.

Les caractéristiques d'une telle approche peuvent se résumer comme suit (Fig. 3.1.) :

i) procédurale et impérative

ii) déterministe

iii) description détaillée de l'algorithme

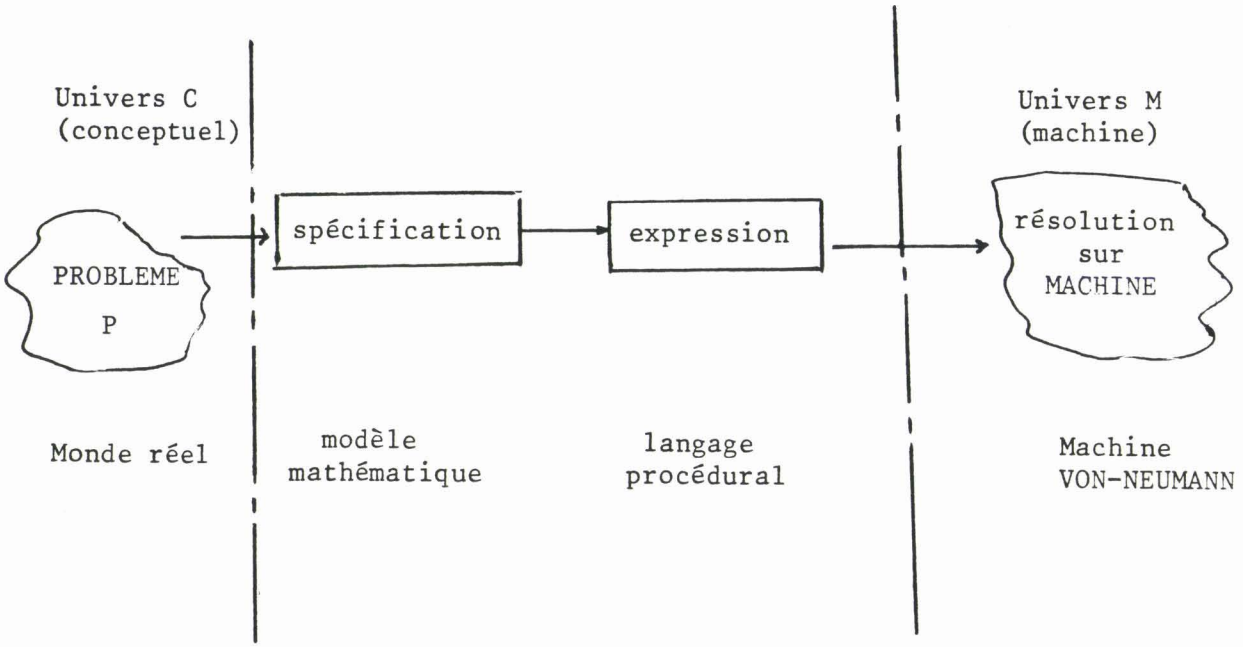


Fig. 3.1. Approche procédurale

3.3.3. Inconvénients des langages procéduraux

Un des inconvénients majeurs des langages procéduraux ou impératifs tels que FORTRAN, PASCAL, etc..., est la présence de l'affectation (assignment). Cette caractéristique fait qu'il est impossible de garantir qu'une fraction de programme en exécution, laissera le reste du programme sans modification (pas de changement d'état).

Ceci montre qu'il est très difficile de raisonner formellement sur ces programmes.

3.3.4. Inconvénients et insuffisances du modèle VON-NEUMANN [BAC 78, SLI 84, SLI 85].

3.3.4.1. Inconvénients

L'inconvénient principal est la notion de mémoire. Celui-ci en induit un deuxième par l'existence d'une relation entre variable et cellule mémoire.

Ainsi la sémantique d'un programme est matérialisée par un ensemble de règles de transition entre états qui sont très complexes dans leur contenu. D'autre part les langages conçus à partir du modèle de VON-NEUMANN sont complexes, volumineux et inflexibles. Leur puissance d'expression limitée est inadéquate pour justifier leur taille et leur coût.

3.3.4.2. Insuffisances

La structure et le fonctionnement des architectures actuelles d'une part, et le type de problèmes rencontrés à l'heure actuelle d'autre part, nous permettent de relever un certain nombre d'insuffisances des systèmes actuels, basés sur les architectures VON-NEUMANN, pour la résolution de ces problèmes.

i) inadéquation entre les caractéristiques des architectures et les domaines de pointe actuels en informatique (intelligence artificielle, robotique, imagerie, etc...). En effet la structure des calculateurs est figée alors que la nature des problèmes se diversifie.

ii) approche algorithmique et déterministe : les calculateurs actuels ne peuvent résoudre, avec des contraintes de coût raisonnables (espace mémoire, temps de calcul), que les types de problèmes décrits sous une forme mathématique et dans un référentiel déterministe.

iii) nécessité de convertir un problème du monde réel en un modèle mathématique. Indépendamment des problèmes posés par la transformation d'un problème d'un univers à un autre, on peut noter que cette conversion crée un décalage d'ordre sémantique entre l'univers conceptuel de l'utilisateur et celui de la machine et ceci est important en graphique car on essaye à travers un système graphique de rendre la "réalité" des objets manipulés.

3.3.4.3. Conclusion

Malgré leurs inconvénients et insuffisances, le modèle VON-NEUMANN et les langages procéduraux qui lui sont associés, resteront toujours utilisés dans le monde informatique. Les architectures de la 5ème génération et les langages de programmation non-procéduraux choisis comme

support de ces architectures, ne remplaceront pas des modèles classiques mais permetteront la mise en place d'une nouvelle approche de résolution d'applications nouvelles. Il faut noter également que les langages non-procéduraux en voie d' implémentation sur les futures architectures de la 5ème génération sont des langages qui existent depuis plus d'une dizaine d'années et parfois depuis plus d'une vingtaine d'années (exemple du langage LISP).

3.3.5. Approche non-procédurale [LEA 74, WEG 76, BAC 78, BOL 83, COL 83, SLI 84, SLI 85]

Ce type d'approche est fondamentalement différent du précédent. Il permet un style de programmation qui est non-procédurale au lieu d'être impérative comme avec les langages conventionnels.

Les langages basés sur cette approche permettent au programmeur de concentrer ses efforts sur la manière de déclarer son problème, plutôt que de penser à sa résolution ou à son exécution. Ce type de langage n'a pu être développé que grâce à la suppression de l'affectation notamment et à la possibilité offerte au programmeur de spécifier ses programmes comme une séquence d'objets tels que des fonctions, des règles, des assertions, des classes, etc...

.3.5.1. Classes de langages non-procéduraux

On distingue généralement trois classes de langages non-procéduraux que l'on peut schématiser comme suit (Fig. 3.2.) [SLI 84]

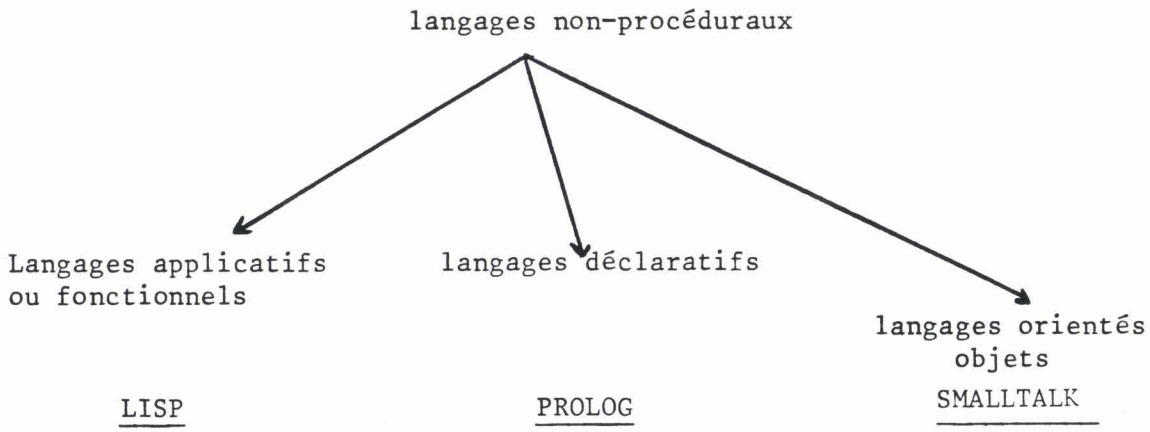


Fig. 3.2. Classes de langages non-procéduraux

Nous avons choisi pour présenter ces classes, trois langages représentatifs qui sont à la base de tous les langages non-procéduraux. Nous détaillerons plus particulièrement les langages applicatifs car le langage choisi pour notre expérimentation appartient à cette catégorie (LE LISP).

3.3.5.1.1. Avantages et inconvénients

Ils peuvent se résumer comme suit :

3.3.5.1.1.1. Avantages

- interprétables
- adaptés à la description de phénomènes ayant un niveau d'abstraction du niveau machine
- adaptés à des manipulations formelles (vérification, preuves de programmes, etc...)

3.3.5.1.1.2. Inconvénients

- difficiles à lire (programmes)
- inappropriés pour des logiciels de large portée
- faiblesse en structures de données (liste)
- nombre restreint de machines support à l'heure actuelle

3.3.5.1.2. Les langages applicatifs [GRE 77, FRI 78, MCC 78-b, BOL 83, HON 83, QUE 83, COI 84-a, COI 84-b, SLI 84, FER 85]

3.3.5.1.2.1. Présentation

Le langage représentatif de cette catégorie est le langage LISP dont nous faisons une présentation sommaire ci-dessous :

Auteur : J. Mc CARTHY (MIT)
Année : 1962
Idées : - spécifications et manipulations d'expressions symboliques
- utilisation du lambda-calcul de CHURCH (λ-calcul)

Types de langage : fonctionnel | récursif | logique | symbolique

Concept de base : λ-calcul

Caractéristiques : intéractif
absence de concepts orientés machine
récursif
formel

But : traitement de listes

programme : suite de fonctions

outils : la liste et la fonction

versions : LISP Pur = LISP 1.5 (Mc CARTHY)
LISP étendu = autres versions (MACLISP, LE-LISP, INTERLISP, ...)

Domaines d'utilisation : traitement de listes
systèmes experts
calcul formel
graphique (expérimentation).

Système lisp : généralement 100 à 200 primitives
(possibilité d'avoir des systèmes à mille primitives)
noyau de base : une trentaine de primitives.

3.3.5.1.2.2. Concepts fondamentaux du langage

Il existe trois concepts fondamentaux dans le langage.

a/ Atome : c'est le plus petit élément du langage

atome ::= symbole | nombre

exemples =

. atomes de type symbole { pixel, ligne, cercle, x, y

. atomes de type numérique { 1986, -5, 3

b/ liste : C'est une composition d'atomes ou de listes entre parenthèses

liste ::= (e1 e2 ... e_n) | ( )

l_i ::= atome | liste

( ) ::= liste vide appelée NIL

exemples =

(Centre rayon origine)
(pixel (x y) cercle)
(segment (x_1 y_1) (x_2 y_2))

c/ fonctions

f ::= fonctions incorporées | fonctions utilisateurs

- fonctions incorporées : Ce sont des fonctions de l'interpréteur mises à la disposition de l'utilisateur

- fonctions utilisateurs : Ce sont des fonctions, comme leur nom l'indique, écrites par un utilisateur en LISP. Ces fonctions sont interprétées par les fonctions d'évaluation du système LISP (EVAL, APPLY).

(de nom de fonction liste paramètres corps)

nom de fonction ::= atome litteral
liste de paramètres ::= $(e_1 \ldots e_n)$ | ( )
Corps ::= $(s_1 \ldots s_n)$

3.3.5.1.2.3. Exemples

Nous présentons ici quelques exemples mettant en valeur les caractéristiques principales de LISP

a/ récursivité : Nous avons choisi pour représenter cette notion, un programme écrit sous forme itérative et récursive.

forme itérative

```
( de somme (liste)
          (let (result 0))
          (while liste
          (setq result (
          (+ result (car liste)))
        (setq result (
             cdr result )))))
```

forme récursive

```
( de somme (liste)
     (cond
         ((null liste)o)
         (t (+(car list)
             (somme(cdr list))))))
```

Ce programme simple permet de faire la somme des éléments d'une liste. On peut remarquer dans la forme itérative un certain nombre de différences par rapport à la forme récursive :

i) utilisation d'une fonction répétitive (while)

ii) utilisation d'une variable locale (result) pour le stockage du résultat

iii) utilisation de l'instruction d'affectation (setq).

b/ interaction :

La fonction interact décrite ci-dessous constitue une boucle d'interaction. Les expressions entrées par l'utilisateur sont lues au clavier, évaluées puis imprimées.

```
( de interact ( )
      (while t
           (setq expr (read))
           (print "= (eval expr))))
```

c/ exemples graphiques

exemple 1 :

```
(de carre (xo yo long)
        (for (xo xo 1 (sub 1 (plus xo long)))
             (afficher écran xo yo)
             (afficher écran xo (sub 1 (plus yo long))))
        (for (yo yo 1(sub 1 (plus yo long )))
             (afficher écran xo yo)
             (afficher écran (sub 1 (plus xo long)) yo )))
```

cette fonction permet d'afficher les pixels d'un carré ayant pour origine les coordonnées (xo yo) et pour longueur (long).

Elle permet d'afficher simultanément les pixels des deux côtés parallèles du carré. L'affichage réel se fait par la primitive graphique (afficher) qui reste à définir (niveau graphique).

exemple 2 :

A partir des fonctions graphiques définies dans le LE-LISP sur les périphériques spécialisés de l'I.N.R.I.A. [CHA 85], nous avons écrit la séquence suivante :

```
(COLORIX 6 0 )                ; efface l'écran visible
(COLORIX 6 ≠ $ F 00 )         ; remplit l'écran de rouge
(COLBOX 10 5 20 8 (COLOR 4 5 6 ))   ; désigne un rectangle
```

```
                                ; d'origine (10 5) de largeur 20, de hau-
                                ; teur 8 et de couleur le résultat de la
                                ; fonction (COLOR 4 5 6).(COLOR 4 5 6)
                                ; fabrique au moyen des trois couleurs pri-
                                ; maires R,V,B (codées sur 4 bits), une
                                ; combinaison codée dur 12 bits.

(COLVECTOR 3 3 35 3 (Color 1 2 3))

                                ; dessine un vecteur de coordonnées
                                ; début (3 3) et de coordonnées fin
                                ; (35 3) et de couleur, le résultat
                                ; de la fonction (COLOR 1 2 3).
```

3.3.5.1.3. Les langages déclaratifs [WAR 77, BOL 83, COL 83, OKE 83, SLI 84]

3.3.5.1.3.1. Présentation

Cette catégorie de langages est représentée par le langage PROLOG (PROgrammation par la LOGique) dont les caractéristiques générales se résument comme suit :

Auteurs : A. COLMERAUER — Université Aix-Marseille
P. ROUSSEL
R. KOWALSKI — Edimbourg

Année : 1972

Idées :- Travaux sur des systèmes de questions/réponses utilisant la langue naturelle
- logique des prédicats

Concepts de base : logique du 1er ordre (ensemble de clauses et de règles d'inférences)

Source : Langage PLANNER développé par C. HEWIT

Caractéristiques : - formel
- naturel
- puissant
- intéractif

Buts : Représenter et utiliser des connaissances sur un domaine
Un domaine est constitué
- d'un ensemble d'objets
- d'un ensemble de relations décrivant les propriétés des objets et leurs interactions

Programme : suite de règles décrivant les objets et les relations

Exécution : Démontrer une ou plusieurs relations à partir d'un ensemble de règles.

Inconvénient : Inefficace quand on connaît la démarche à suivre pour trouver la solution d'un problème (algorithme).

Outil : notion d'arbre (éventuellement infini)

Primitive fondamentale : l'unification

Domaines d'utilisation :
- communication en langue naturelle
- calcul formel
- écriture de compilateurs
- banque de données
- C.A.O.
- systèmes experts

3.3.5.1.3.2. Exemple

Analyseur syntaxique [COL 83]

L'exemple présenté ici concerne un analyseur de la partie expressions d'un langage simple. L'un des aspects importants de ce programme est qu'il est écrit sous une forme qui s'apparente beaucoup à la norme Backus-Naur (BNF).

Il permet de reconstituer à partir d'une phrase du langage (ici une expression arithmétique entière) sa structure selon la grammaire d'écriture.

"Grammaire des expressions"

```
expression (e,< x, y >) → somme (e, < x, y >) ;
somme (e, < x, y > ) → produit (p, < x, z, >) reste-de-somme
      (p, e < z, y >) ;
produit (p, < x, y >) → primaire (f, < x, z >) reste- de-produit
       (f, p, < z, y > ) ;
primaire (n, < x, y >) → mot (n, < x, y >) entier (n) ;
```

```
primaire (e, < x, y >) → mot ("(", < x, z >) expression (e,< z, t >)
                         mot (")", < t, y >) ;

reste-de-somme (p, e, <x, y > ) → mot (o, < x, z >) op- add <o,o-a)
               produit (p', <z, t >) reste-de-somme (< o-a, p,p'>, e,
               < t, y > ) ;

reste-de-somme (e, e < x, x >) → ;

reste-de-produit (f, p, < x, y >) → mot (o, < x, z >) op-mul (o,o-m)
        primaire (f', < z, t >) reste-de-produit (<o-m, f,f'>, p,
                 < t, y >) ;
reste-de-produit (f, f, <x, x >) → ;
mot (< a, x, x >) → ;
op-add (" + ", add) → ;
op-add ("-", sub) → ;
op-mul (" * "  mul) → ;
```

"lecture"

```
lire (nil) → car-après (".") / in-car'(".") ;
lire (a.b) → in-ident (a) / lire (b) ;
lire (a.b) → in-entier (a) / lire (b) ;
lire (a.b) → in-car' (a) lire (b) ;
```

"lancement"

```
en route →

      exm ("l'expression")
      lire (n)
      analyse (p,e)
      val (e,f)
      exm ("    vaut    ") ex (f) ;
      analyse (p,e) → expression (e, < p, nil >) / ;
      analyse (p,e) → exm ("... est incorrecte") impasse ;
```

3.3.5.1.4. Les langages orientés objets [SHO 79, COI 82, BOL 83, TAK 83, BEZ 84, COI 84-a, COI 84-b, SLI 84]

3.3.5.1.4.1. Introduction

Ils sont également appelés langages de communication par messages ou langages d'acteurs. Les principales caractéristiques de ces langages peuvent être résumées comme suit :

i) conception de la programmation reposant sur le concept d'objet (héritage de SIMULA 67), alors que la programmation classique repose sur le concept de procédure.

ii) suppression de toute distinction entre programmes et données, alors que la plupart des langages de programmation usuels imposent une coupure radicale entre information manipulée et code manipulant.

iii) chaque entité du langage (nom, fonction) est considérée comme un objet.

iv) utilisation d'une seule structure de contrôle (la transmission) et une seule structure de données (l'objet)

Le langage le plus représentatif de cette catégorie est le langage SMALLTALK dont nous faisons une présentation sommaire dans le paragraphe suivant.

3.3.5.1.4.2. Le langage SMALLTALK [GOL 83]

Auteurs : A. KAY
C. HEWIT

Année : 1970

Source : SIMULA 67, LISP

Idées : - créer des objets issus de classes
- manipuler des objets et des classes de manière temporelle

Concepts : - les objets
- les classes qui représentent une composition d'objets ainsi que des règles de comportement envers les messages qui leur sont communiqués.

Programme : - description de classes
- description de comportements

Caractéristiques :

- environnement interactif utilisant des moyens matériels et logiciels très riches avec en particulier des écrans à très haute résolution partageables dynamiquement en un grand nombre de fenêtres.

- hiérarchie des modules (classes) et d'héritage qui jouent un rôle fondamental dans l'écriture de logiciels réutilisables.

- simplification de la syntaxe LISP, tout en élargissant la gamme et la complexité des objets manipulés.

- introduction de la classe WINDOW qui est le premier maillon d'un environnement de programmation créatif. Elle permet de gérer un écran vidéo à la manière d'une table de travail.

- constitue un ensemble d'objets rémanents et plurifonctionnels dotés en outre d'une base de données, c'est-à-dire d'un environnement transportable par ces objets.

- extensible et applicatif.

3.3.5.1.4.3. Exemples [COI 82]

a/ création d'une classe définissant une forme compteur [SMALLTALK-76]

```
CLASS new ; title  COMPTEUR ; fields  (cpt) ; as follows !

      incr [cpt ← cpt + 1]
      decr [cpt ← cpt - 1]
      raz  [cpt ←  0 ] !
```

On définit ainsi un objet-registre capable de reconnaître les trois opérations de remise à zéro(raz), d'incrémentation et de décrémentation (incr et decr).

b/ La transmission

classe prédéfinie VECTOR

```
↑ joe !                    , receveur → joe
# (1233567)
↑
↑ joe is !                 , receveur → joe
# VECTOR                   , sélecteur unaire → is
↑ vector selectors !       , selector livre l'ensemble
                             des méthodes connues d'une
                             classe,
# (is PRINT EVAL REVERSE LENGTH # + ← selectors)
↑ joe reverse !            , sélecteur unaire → reverse,
                           , pas d'argument,
#(7 6 5 3 3 2 1)
↑ joe + joe !              , selecteur binaire → +
# (1233567 1233567)        , un argument joe,
```

c/ récursivité [GOL 83]

```
factorial
self = 0 ifTrue : [↑1]
self < 0
        iftrue : [self error = 'factorial invalid']
        ifFalse : [↑self * (self - 1) factorial]
```

d/ Graphique [GOL 83]

Prenons l'exemple classique de tracé de polygones. Un carré peut être créé par l'expression SMALLTALK suivante :

```
4 times Repeat : [bic go : 100. bic turn : 90]
```

Indépendamment de la syntaxe SMALLTALK on spécifie qu'on répète 4 fois le tracé d'un segment de longueur 100 avec un angle de rotation de 90°.

L'expression suivante crée une figure de type polygone en calculant l'angle de rotation comme étant une fonction du nombre de côtés du pôlygone. Si nSides est le nombre de côtés du polygone désiré, alors :

nSides times Repeat : [bic go : 100. bic turn : 360 // nSides]

permet de tracer le polygone.

On peut ainsi créer une classe de polygones où les instances réfèrent le nombre de côtés et la longueur de chacun des côtés. En plus chaque polygone dispose de sa propre Classe Pen permettant l'affichage.

La méthode de tracé de polygone suivant le principe ci-dessus, se décrit en SMALLTALK comme suit [GOL 83] :

```
* nom classe              polygon
* super classe            objet
* noms des instances      drawing Pen
                          nSides
                          length

* méthodes de la classe

    création d'une instance
        new
          ↑ Super new default

* méthodes des instances

    drawing
      draw
          drawingPen black
          nSides times Repeat : [drawing Pen go : length : turn : 360 //
          nSides ]

* accès

     length : n
        length ← n
        Sides : n
        nSides ← n
```

* particularité de la classe

default

```
drawing Pen ← Pen new
self length : 100
self Sides : 4
```

Ainsi un polygone peut être créé comme une séquence de polygones affichés par l'évaluation de l'expression suivante, qui permet de tracer 8 polygones imbriqués dont le nombre de côtés varie de 3 à 10.

```
poly ← polygon new
3 to : 10 do : [: sides | poly sides : sides. poly draw ]
```

L'évaluation de cette expression donne la figure suivante (fig. 3.3.)

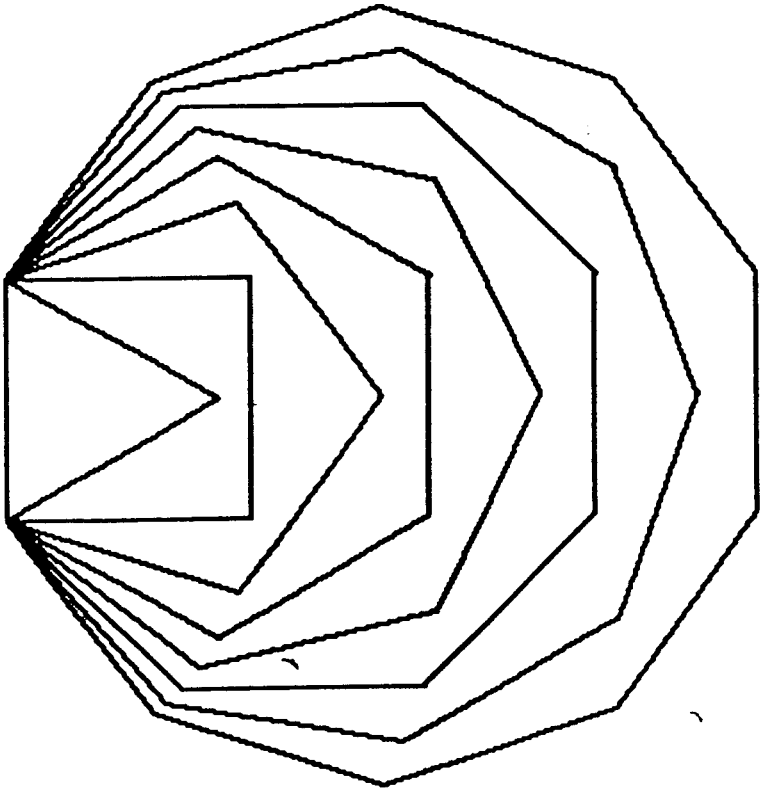


Fig. 3.3 Tracé de polygones imbriqués en SMALLTALK

3.4. Les langages applicatifs : aspects conceptuels

3.4.1. Introduction

Les langages applicatifs ou fonctionnels sont des langages basés sur l'évaluation d'expressions, avec effet de bord, formées à partir de constantes, paramètres formels, fonctions et opérateurs fonctionnels (formes combinées pour les fonctions, telles que la composition).

Les exemples les plus connus de modèles applicatifs sont le LAMBDA-CALCUL de CHURCH, les systèmes combinatoires de CURRY, les systèmes LISP et les systèmes de programmation fonctionnelle [BAC 78].

3.4.2. Caractéristiques [REY 70, BAC 78, MCC 78-b, LEV 79, BOL 83, HON 83, COI 84-b]

A la différence des langages de programmation usuels, les langages basés sur les systèmes applicatifs n'utilisent pas de notion de mémoire (identité entre variable et cellule mémoire). C'est d'ailleurs l'une des raisons majeures qui a fait que de tels langages n'ont pas connu une large utilisation pour la conception de machines. Leurs principales caractéristiques sont :

i) la suppression de la distinction entre programme et données

ii) la banalisation des notions de pointeurs, d'appel par valeur et de récursivité.

iii) l'utilisation de la programmation fonctionnelle qui introduit une nouvelle technique de programmation dans laquelle les calculs sont effectués sous une forme quasi-parallèle.

iv) l'auto-construction de langages à partir d'un ensemble réduit de primitives (fonctions).

v) utilisent comme opération de base, l'opération de substitution du LAMBDA-CALCUL.

3.4.3. Comparaison avec les langages de type VON-NEUMANN [BAC 78, SLI 85]

Nous allons essayer de mettre en évidence les différences entre programmes applicatifs et programmes procéduraux en faisant ressortir les propriétés de chaque catégorie de programmes.

Programme procédural	Programme applicatif
1°/ ses instructions opèrent sur des états "invisibles" (transparents à l'utilisateur) suivant des règles complexes	- opère seulement sur ses arguments. Il n'y a pas ici d'états cachés ou de règles de transition complexes. On trouve que deux catégories de règles : l'une pour appliquer une fonction sur ses arguments et l'autre pour obtenir la fonction dénotée par une forme fonctionnelle telle que la composition
2°/ non hiérarchique	- hiérarchique (bâti sur des fonctions simples)
3°/ sémantique informelle et procédurale	- sémantique formelle
4°/ présente des contraintes mémoire (référence et accès à la mémoire à chaque instant)	- abscence de notion de mémoire
5°/ dynamique et répétitif : sa compréhension est liée à son exécution mentale	- statique et non répétitif en ce sens que sa structure est très appropriée pour sa compréhension sans exécution mentale.
6°/ opère sur des variables temporelles	- opère sur un ensemble d'unités conceptuelles et non sur des variables mémoire
7°/ Données incorporées avec le programme (mélange et distinction entre données et instructions).	- les données ne sont pas incorporées dans le programme, ce dernier est général.
8°/ Développé à partir de langages dans lesquels variable = cellule mémoire. Ces langages sont volumineux, complexes et inflexibles. Leur puissance d'expression limitée est inadéquate pour justifier leur taille et leur coût.	- développé à partir de langages offrant une grande liberté pour modifier le texte du programme et échantillonner son exécution sans avoir à suivre une discipline rigide d'édition/compilation/test associée avec les compilateurs traditionnels.

Comparaison entre programme procédural et programme applicatif

3.4.2. Avantages et Inconvénients

Les langages applicatifs retiennent de plus en plus l'attention actuellement à cause de leurs nombreux avantages théoriques et pratiques. On rappellera ici que ces langages ont plus de vingt années d'existence (exemple du langage LISP) et que ce n'est que ces dernières années, par l'émergence d'applications nouvelles, que leurs possibilités sont redécouvertes et mises en pratique.

Les avantages des langages applicatifs ayant été mis en évidence tout au long de ce chapitre, nous insisterons sur l'un des plus important, à savoir leur très bonne adaptabilité à décrire des phénomènes ayant un niveau d'abstraction du niveau machine.

Considérons par exemple l'expression fonctionnelle suivante :

$$(f[g[x], h[y]]) \qquad (1)$$

qui spécifie une application des fonctions g et h aux arguments x et y, puis l'application de la fonction f au résultat des précédentes applications.

L'expression (1) spécifie exactement ce dont on a besoin pour calculer la valeur désirée, mais ne précise aucune contrainte sur ces données, sur le contrôle, sur le processeur, ou sur toute autre structure utilisée pour le faire.

Par exemple :

- g[x] et h [y] sont-elles évaluées séquentiellement ou en parallèle ?
- dans quelles structures de données leurs valeurs sont-elles stockées?

Autant de questions donc, pour lesquelles l'expression ne fournit aucune réponse, ce qui a pour effet d'éliminer toute contrainte au niveau de l'expression et permet de différer les décisions à prendre pour répondre à de telles questions.

Comme inconvénients des langages applicatifs on peut citer les points suivants :

i) mauvaise clarté des programmes. En effet les programmes applicatifs ont la réputation d'être difficiles à lire (à écrire aussi). Des efforts sont entrepris dans ce sens pour les rendre plus clairs et plus agréables à lire.

ii) inappropriés généralement pour des logiciels de large portée

iii) gros consommateurs de place mémoire et d'unité centrale, suivant les systèmes sur lesquels ils sont implémentés.

CHAPITRE 4 : UN ESSAI D'IMPLEMENTATION EN LE-LISP

CHAPITRE 4

UN ESSAI D'IMPLEMENTATION EN LISP

****
****
****

4.1. Introduction

A travers les paragraphes de ce quatrième chapitre, nous mettrons en évidence notre contribution dans l'utilisation des langages applicatifs en graphique à travers les quelques expériences que nous avons réalisées.

Nous rappelons ici que l'objectif que nous nous étions fixé ne concernait nullement la production d'images de bonne qualité, mais plutôt d'apporter des éléments de réponse à la question que nous nous étions posés au début de ce travail : "Quel serait l'apport des langages applicatifs dans le domaine du graphique".

Il est évident que les expériences que nous avons réalisées, dans un laps de temps relativement court, ne permettent pas de fournir une réponse formelle à une telle question, mais peuvent servir d'éléments d'appréciation pour des recherches en algorithmique graphique récursive, et en représentation des images sous forme de listes.

4.2. Le langage LISP

4.2.1. Choix du langage

Le choix d'un tel langage répond à deux types de considérations.

4.2.1.1. Considérations d'ordre général

i) utilisation d'un langage non-procédural

ii) utilisation d'un langage non-procédural renfermant quelques concepts se rapprochant du graphique. Dans ce domaine beaucoup de problèmes se caractérisent par leur aspect récursif. L'étude des langages non-procéduraux nous a permis de fixer notre choix sur les langages applicatifs qui sont récursifs par excellence, et en particulier le langage le plus représentatif, à savoir le LISP.

4.2.1.2. Considérations d'ordre technique

i) le langage LISP est un langage non typé (ou très faiblement typé), et toutes les vérifications sont faites dynamiquement à l'éxécution. Ceci permet de ne pas avoir à traiter d'importantes déclarations de types, comme en PASCAL par exemple.

ii) un autre intérêt du non-typage de LISP est celui de pouvoir garder une structure uniforme pour les fonctions et les listes.

iii) LISP est maintenant un langage répandu dans le monde entier, qui bénéficie d'implémentations efficaces, comprenant notamment un compilateur, indépendamment de l'existence sur le marché de machines-LISP. Du strict point de vue de l'efficacité en temps, il n'y a donc plus de raisons de lui préférer un langage impératif plus classique.

iv) LISP est également efficace du point de vue de la gestion de la mémoire puisque son mécanisme de "ramasse-miettes" (garbage-collector) permet de récupérer automatiquement les cellules mémoires allouées et libérées, sans interventions explicites de la part de celui qui écrit un programme. De plus la structure de base de LISP étant la liste, les gros attributs (consommateurs de mémoire), comme une mémoire de pixels par exemple, seront donc naturellement implantés par des listes, ou des morceaux de listes. Cette efficacité provient du fait que LISP manipule toutes ses données par l'intermédiaire de pointeurs, la copie de données se ramène donc à la copie de pointeurs qui occupent généralement moins de place qu'une duplication des données.

v) La disponibilité au laboratoire d'informatique de l'Université de Lille 1, sous le système MULTICS, de deux versions de LISP, le MACLISP [MAC 84-b] et le LE-LISP [CHA 85]

4.3. Historique de LISP [MCC 78-b]

Le développement du langage LISP s'est fait sur deux périodes :

i) la période 1956-1958 au cours de laquelle un certain nombre d'idées ont été développées. Ces idées puisent leur source essentiellement dans le type de problèmes que cherchait à résoudre Mc CARTHY (problèmes de déduction) et du type d'information manipulée, à savoir la notion d'information symbolique et non numérique.

Une considération mathématique qui influencera LISP, consistait à formuler les programmes comme des expressions applicatives construites à partir de variables et de constantes utilisant des fonctions. Mc CARTHY considérait qu'il était important de formuler des expressions en respectant les lois mathématiques usuelles permettant le remplacement (substitution) des expressions par des expressions donnant la même valeur. Cette démarche offrait la possibilité de réaliser des preuves de programmes en utilisant les méthodes mathématiques ordinaires.

ii) la période 1958-1962 au cours de laquelle ont été réalisées les implémentations et les applications du langage.

Comme langage de programmation, LISP se caractérise par les idées suivantes :

a) utilisation du concept symbole au lieu de celui de nombre

b) représentation des informations par une structure de listes en mémoire

c) représentation externe de l'information par des listes à multi-niveaux

d) existence d'un noyau de constructeurs et de sélectionneurs de listes exprimés sous forme de fonctions

e) utilisation de la composition pour l'expression de fonctions complexes

f) utilisation d'expressions conditionnelles pour réaliser des branchements à l'intérieur des fonctions.

g) représentation unique d'un programme et de ses données

h) utilisation de la fonction EVAL à la fois comme définition formelle du langage et comme interpréteur.

4.4. Environnement de programmation

4.4.1. Le langage LE-LISP

Le LE-LISP est un système LISP developpé à l'INRIA. Il est composé :

i) d'un interprète
ii) d'un compilateur
iii) d'outils de développement

CHAILLOUX dans [CHA 85] le désigne comme le fils naturel du MACLISP avec lequel nous avons débuté nos premières expériences de transcription d'algorithmes graphiques. C'est un système facilement portable écrit dans le langage de la machine virtuelle LLM3 [CHA 84-a], indépendant des systèmes d'exploitation, qui fonctionne sur le DPS 8 MULTICS de l'Université de LILLE 1. Tous les types d'objets LISP sont gérés dynamiquement ; nombres flottants, symboles, chaînes de caractères, vecteurs de S-expressions et cellules de listes. Il utilise un tas pour gérer dynamiquement les objets de taille variable (chaînes et vecteurs),

L'interprète LE-LISP permet de manipuler des objets appelés S-expressions (pour symbolic expression). Ces objets sont :

i) les objets atomiques (symboles, nombres, chaînes)
ii) les objets composites (listes, vecteurs).

LE-LISP permet également de définir de nouveaux types qu'on appelle les types étendus.

4.4.1.1. Les objets atomiques

Ils jouent le rôle d'identificateurs et servent à nommer les variables, les fonctions et les échappements.

Un symbole est représenté dans l'interprète par un pointeur sur un descriptif stocké dans une zone spéciale de la mémoire et il est constitué des 9 propriétés naturelles suivantes (fig. 4.1.) :

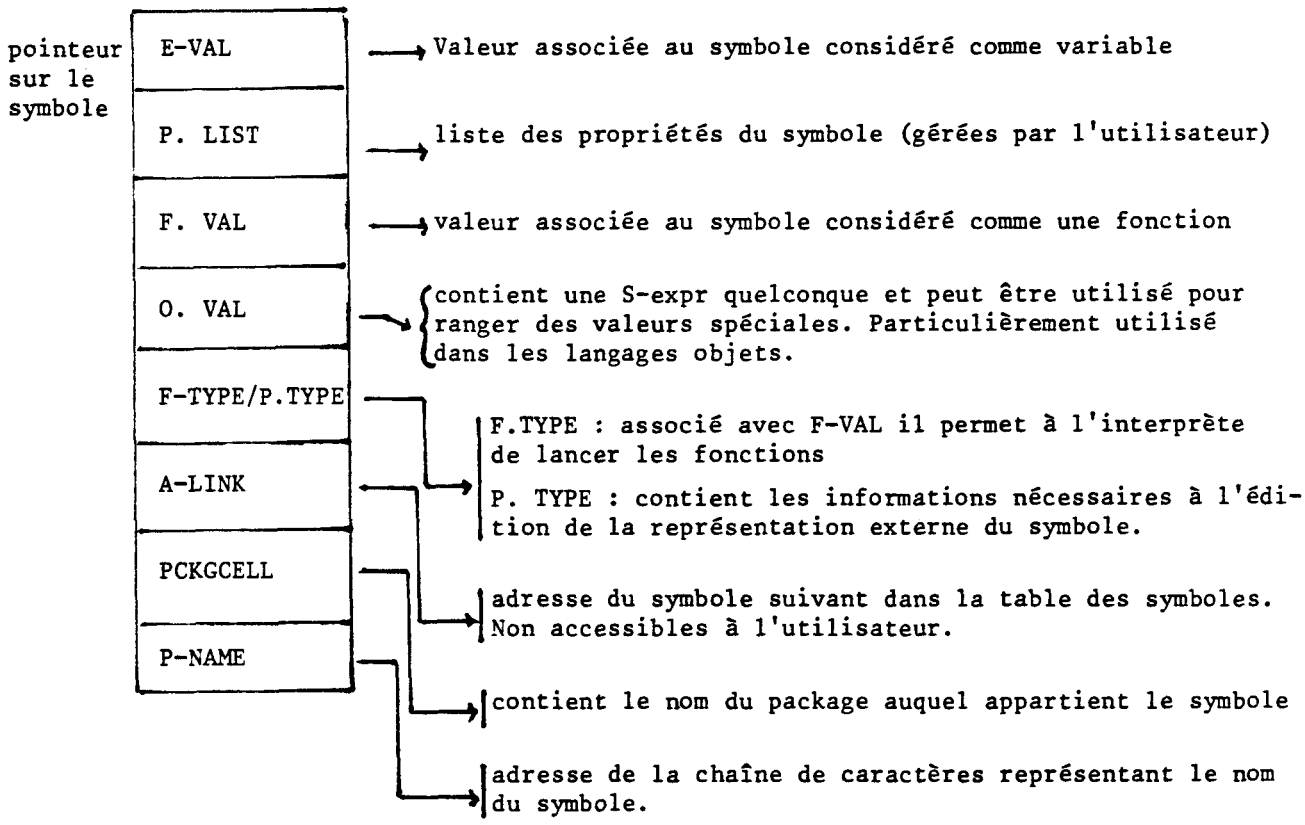


Fig. 4.1. Représentation d'un symbole.

La représentation d'une S-expression est faite au moyen d'un pointeur vers sa ou ses valeurs. L'accès se fait donc toujours par indirection. Ainsi une cellule mémoire pourra contenir n'importe quel objet et le même objet peut être contenu dans différentes cellules mémoires.

4.4.1.2. Les objets composites

Ils sont formés au moyen d'autres objets LISP. L'exemple classique est celui des listes. Un élément de liste est stocké dans une cellule de liste constituée d'un doublet de pointeurs (un CONS) dont la partie gauche (le CAR) contient l'élément et la partie droite (le CDR) contient un pointeur sur l'élément suivant ou bien un marqueur spécial indiquant la fin de la liste (Fig. 4.2.) :

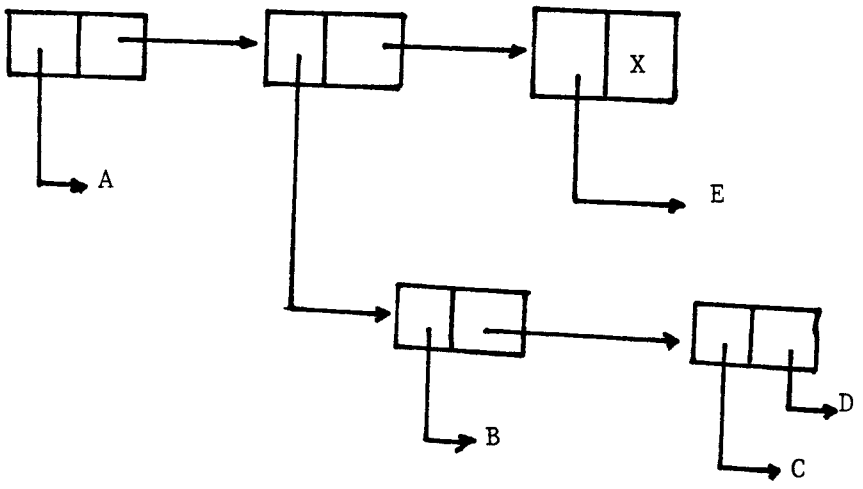


Fig. 4.2. Représentation mémoire de la liste (A(BC.D)E)
(remarquons l'analogie avec les arborescences)

4.4.1.3. Les fonctions

Le LE-LISP possède 6 familles de fonctions traitées différemment par l'interprète :

i) les SUBR et les FSUBR écrites en langage machine LLM3

ii) les EXPR, les FEXPR, les MACRO et les DMACRO écrites en LISP

4.4.1.3.1. Les SUBR

Ce sont des fonctions écrites en langage machine, extrêmement rapides à l'éxécution et résidentes dans l'interprète dès son lancement (fonctions prédéfinies) ou après compilation (les fonctions compilées). Ces fonctions possèdent des arguments qui sont toujours évalués. On peut obtenir des SUBR après compilation des EXPR. Ce type de fonctions (SUBR) sont très nombreuses dans l'interprète et représentent un total de 400 à 500 en fonction du système utilisé.

Exemples : APPLY, CAR, CDR, COMPILER, MAKEARRAY, TYCURSOR

4.4.1.3.2. Les FSUBR

Elles sont également écrites en langage machine et résidentes dans l'interprète dès son lancement. Elles possèdent des arguments en nombre variable qui ne sont jamais évalués par l'interprète, mais le cas échéant par la fonction elle-même. Ce sont des fonctions spéciales utilisées comme fonctions de contrôle de l'interprète ou comme fonctions de manipulation de noms. Elles sont peu nombreuses dans un système LISP. On peut augmenter le nombre de FSUBR en écrivant de nouvelles en langage machine LLM3 ou en compilant des FEXPR.

Exemples : COMPILE, DE, GO, LAMBDA, PRECOMPILE, PROG

4.4.1.3.3. Les EXPR

Ce sont des fonctions écrites en LISP et interprétées par les fonctions d'évaluation (EVAL, APPLY, FUNCALL). Elles possèdent à leur définition un certain nombre de paramètres représentés par une liste (ou un arbre) de variables et un corps. L'appel de ce type de fonction se fait par valeur. La description d'une fonction de ce type se fait en utilisant une liste, appelée LAMBDA-expression

```
Exemples : (LAMBDA (x y ) ( CONS (CAR x ) ( CDR y )))
           ((LAMBDA x x ) (1 + 1) (1 + 2) (1 +3 ))
            => ( 2 3 4).
```

exemples de fonctions : HANOI, PEPEFILE, PEPEND,

4.4.1.3.4. Les FEXPR

Identiques aux EXPR, elles n'y diffèrent que par le mode de liaison : c'est l'ensemble des arguments non évalués, c'est-à-dire le CDR de la forme elle-même, qui est lié à la liste de paramètres en utilisant une procédure de liaison récursive comme pour les EXPR. Les fonctions de ce type sont décrites en utilisant une liste, appelée FLAMBDA-expression.

Exemple : (FLAMBDA (var val) (SET var (EVAL val)))

Exemples : DEBUG, UNTRACE

4.4.1.3.5. Les MACRO

Comme les EXPR et les FEXPR, elles sont écrites en LISP et possèdent des paramètres en nombre variable rangés dans une liste ainsi qu'un corps. Elles sont décrites en utilisant une liste appelée MLAMBDA-expression.

(MLAMBDA (n var) (LIST 'SETQ var (LIST 'CDR var)))

L'évaluation d'une forme MACRO se fait en deux temps :

i) évaluation de la fonction associée à cette MACRO avec toute la forme(non évaluée) comme argument

ii) re-évaluation de la valeur retournée de la première évaluation.

Ainsi il est possible de réaliser des modifications physiques de la forme elle-même à la première évaluation de la MACRO.

Exemple : ERRSET

4.4.1.3.6. Les DMACRO

Leur définition se fait à l'aide de la fonction DMD. Leur évaluation se distingue des MACRO par :

i) l'utilisation du CDR de la forme (non évalué) comme argument à la manière des FEXPR

ii) le remplacement physique de la forme toute entière, au retour de la première évaluation, par la valeur retournée au moyen de la fonction DISPLACE (d'ou le nom de DMACRO → Displace MACRO).

4.5. Choix d'une méthode d'expérimentation

4.5.1. Introduction

L'utilisation d'un nouveau langage en graphique peut se faire à notre avis selon trois méthodes :

i) la transcription d'algorithmes graphiques dans ce langage

ii) l'exploitation des concepts du nouveau langage, pour la spécification d'algorithmes graphiques.

iii) l'intégration d'un langage graphique, en l'occurence la norme internationale GKS, à ce nouveau langage.

4.5.2. Transcription d'algorithmes

C'est une méthode triviale qui consiste à transcrire dans le nouveau langage, les algorithmes classiques du graphique (tracé de segment, de cercle, primitives de transformations, remplissage, etc...). Nous avons utilisé cette méthode pour nos premières expérimentations qui nous ont permis de nous familiariser avec les langages MACLISP et LE-LISP. C'est une méthode inintéressante que nous avons abandonné car elle ne nous permettait pas d'exploiter les fondements du langage notamment en matière de récursivité. D'autre part elle présente un certain nombre de contraintes dans la mesure où elle nous astreint à respecter l'esprit de programmation des algorithmes classiques (méthode itérative, respect des structures de données, etc...). Elle nous a permis de mettre en évidence un certain nombre de problèmes de programmation dûs principalement à la différence existante entre les langages de programmation procéduraux et non-procéduraux (PASCAL, MACLISP et LE-LISP). Nous avons également abouti à des structures de programmes incompréhensibles dans leur forme.

4.5.3. Spécifications d'algorithmes graphiques

L'idée de base de cette méthode est d'essayer d'exploiter les caractéristiques du nouveau langage pour l'expression d'algorithmes graphiques. Dans notre cas, il s'agit d'exploiter le concept de récursivité du LE-LISP pour spécifier certains types d'algorithmes. C'est la méthode que nous avons choisi pour écrire certains algorithmes en LE-LISP que nous détaillerons dans le paragraphe suivant. Nous avons pu ainsi aborder certains problèmes du graphique suivant une approche différente de l'approche classique qui est une approche itérative ou procédurale. Elle répond d'autre part aux objec-

tifs que nous nous étions fixés au départ, à savoir l'utilisation des concepts des langages applicatifs.

4.5.4. Intégration d'un langage graphique

C'est une méthode beaucoup plus générale car elle répond à un souci de standardisation pour l'utilisation de la norme G.K.S. Il s'agit ici de tenter les mêmes expériences qui ont été réalisées avec les langages FORTRAN, PASCAL et plus récemment C et ADA. C'est une voie de recherche très intéressante qui peut être un des prolongements de notre travail, après une bonne maîtrise du langage LE-lISP par la spécification d'algorithmes récursifs.

4.6. Réalisation pratique

4.6.1. Introduction

Dans ce paragraphe, nous essayerons de mettre en évidence à travers les quelques algorithmes que nous avons choisi pour les besoins de notre expérimentation, les apports des langages applicatifs, et en particulier le LE-LISP, au domaine graphique.

L'étude de ces algorithmes ainsi que l'exploitation des résultats que nous avons obtenu , permettrons notammant de répondre à la question que nous nous étions posé en entammant ces travaux, à savoir :

"Quels seraient les apports des langages applicatifs au graphique, en matière d'expressions d'algorithmes et de structures de données".

L'orientation de notre expérimentation a porté sur les aspects algorithmique et structures de données en synthèse d'images. Pour cela nous avons choisi deux axes d'expérimentation :

i) un axe algorithmique avec l'utilisation de deux algorithmes élémentaires (tracé et remplissage)

ii) un axe modélisation avec les arbres quaternaires.

4.6.2. Algorithme de tracé de segment

Le segment de droite est la primitive graphique la plus usitée pour la génération de dessins ou d'images. Un segment est caractérisé par les coordonnées de ses extrêmités que nous noterons (x_d y_d) pour le début et (x_f y_f) pour la fin.

Divers algorithmes existent pour tracer un segment [LUC 77, MER 84,HEG 85-b] et diffèrent les uns des autres par :

i) la méthode de construction (pente, dichotomie, analyse différentielle, propagation, ...)

ii) le type de valeurs utilisées (entières, réelles)

iii) le type d'algorithme (incrémental, parallèle, dichotomique, ...)

En conformité avec l'orientation de nos travaux, nous avons choisi la méthode dichotomique, qui nous permettra de spécifier un algorithme récursif de tracé de segment.

4.6.2.1. Présentation

1°/ Soit (a ← (x_d y_d)) et (b ← (x_f y_f)) les deux extrêmités d'un segment S à tracer.

2°/ Le milieu (m ← (diviser (plus x_d x_f)2) (diviser (plus y_d y_f)2)) appartient à S.

3°/ En faisant le même raisonnement pour les sous-segments (am) et (mb) nous obtenons quatre sous-segments de S.

4°/ En poursuivant le processus de décomposition, le segment S est entièrement tracé par une méthode récursive.

Partant de cette méthode, nous pouvons faire une tentative d'expression en LE-LISP, d'une fonction récursive de tracé de segment.

4.6.2.2. Algorithme en LE-LISP

L'architecture de notre algorithme est composée de 4 fonctions LE-LISP dont une récursive (Fig. 4.3.).

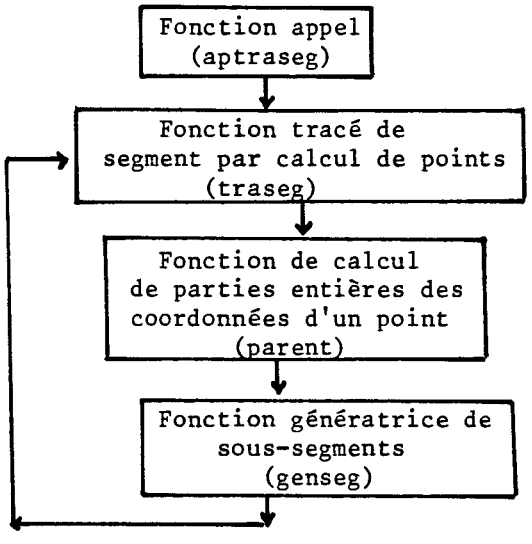


Fig. 4.3. Architecture de l'algorithme récursif de tracé de segment

a/ Fonction aptraseg (de aptraseg (n)

argument : n → taille de l'espace écran

fonctions : * création d'un tableau image de l'écran
* transformation des coordonnées utilisateur (espace orthonormé) en coordonnées tableau.
* affichage des deux extrêmités du segment (mise à 1 de leur pixel correspondant)
* appel de la fonction traseg

Structures de données : * un tableau image de l'écran
* une liste comprenant les coordonées du segment à tracer

b/ Fonction traseg : (de traseg (liseg écran)

arguments : (liseg) → liste des sous-segments à traiter
(écran) → tableau image de l'écran servant de support pour l'affichage

structures de données : * liste des sous-segments à traiter (liseg)
* liste des coordonnées entières du segment en cours (coordent)
* tableau image de l'écran

Fonctions : - Affiche tous les points situés entre les deux extrémités du segment initial par méthode dichotomique
- elle s'appelle récursivement avec comme argument la liste des sous-segments à traiter

c/ Fonction parent : (de parent (coordent)

arguments : (coordent) → liste de coordonnées à transformer en coordonnées entières

Structure de données : * liste comprenant les coordonnées début, fin et milieu du segment en cours (ces valeurs sont de type réel)

Fonction : - transformation d'une valeur réelle en valeur entière.

d/ Fonction genseg ; (de genseg (liseg xm ym)

arguments : (liseg) → liste des sous-segments à traiter

(xm ym) → coordonnées milieu du segment en cours

Fonction : génère à partir des coordonnées début, fin et milieu du sous-segment courant, deux nouveaux sous-segments.

4.6.2.3. Conclusion

La réalisation en LE-LISP de fonctions de tracé de segment par méthode dichotomique et leur expérimentation nous permet de faire les remarques suivantes :

i) Si la méthode de tracé (division par 2 d'un segment) est récursive, sa réalisation en fonctions LE-LISP ne permet pas d'exprimer de façon naturelle cette récursivité. En effet cette dernière n'est pas très apparente et on peut se demander s'il ne faudrait pas lui préférer d'autres versions d'algorithmes. Nous pensons notamment à un algorithme parallèle.

ii) La deuxième remarque, c'est que nous avons été obligé d'utiliser des listes, et d'opérer sur ces listes toute une "chirurgie" pour gérer les coordonnées des segments produits par décomposition. Ceci influe énormément sur le temps de tracé. Par exemple on n'arrive pas à tracer sur un écran 256x256, le segment diagonale, parce que la pile du système LE-LISP est saturée.

On peut donc signaler que l'exemple du tracé de segment montre, à notre avis, que l'écriture systématique sous forme récursive d'algorithmes graphiques n'est pas toujours évidente et peut compromettre leur efficacité. Il y a donc lieu de faire une classification d'algorithmes récursivables et non récursivables en tenant compte notamment de trois critères de complexité : écriture, occupation mémoire, temps de calcul.

4.6.3. Algorithme de remplissage [FOL 82, HEG 85-b]

4.6.3.1. Présentation

Le remplissage de taches est une opération qui consiste à remplir une zone du plan dont la frontière est définie soit par un contour polygonal ou autre dans l'espace utilisateur (Fig. 4.4.), soit par un contour préinscrit point par point dans une mémoire d'image représentant l'espace écran (Fig. 4.5)

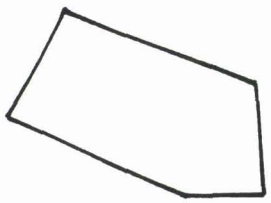


Fig. 4.4 Contour polygonal

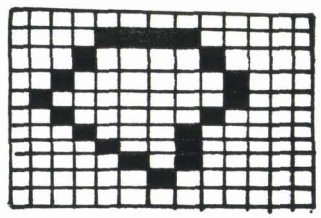


Fig. 4.5 Contour défini point par point

Généralement, l'algorithme de remplissage se décompose en deux phases :

i) préparation du contour : constitution d'une liste d'arêtes lorsque l'on raisonne dans l'espace utilisateur, ou inscription et codage du contour dans une matrice image représentant la surface de visualisation.

ii) remplissage proprement dit avec la couleur désirée

La procédure la plus simple, qui nous permet d'avoir un processus récursif, consiste à s'inspirer du célèbre jeu de la vie de CONWAY.

Le principe d'un tel processus est le suivant :

A partir d'un pixel intérieur au contour (choisi préalablement), auquel on a affecté la couleur que doit avoir la tâche, on fait une propagation de proche en proche de cette couleur. Un pixel transmet cette nouvelle couleur à tous les pixels qui lui sont 4-connectés(les 4 voisins immédiats). La propagation se fait jusqu'à ce que l'on rencontre un pixel ayant une couleur différente de celle du pixel de départ (Fig. 4.6).

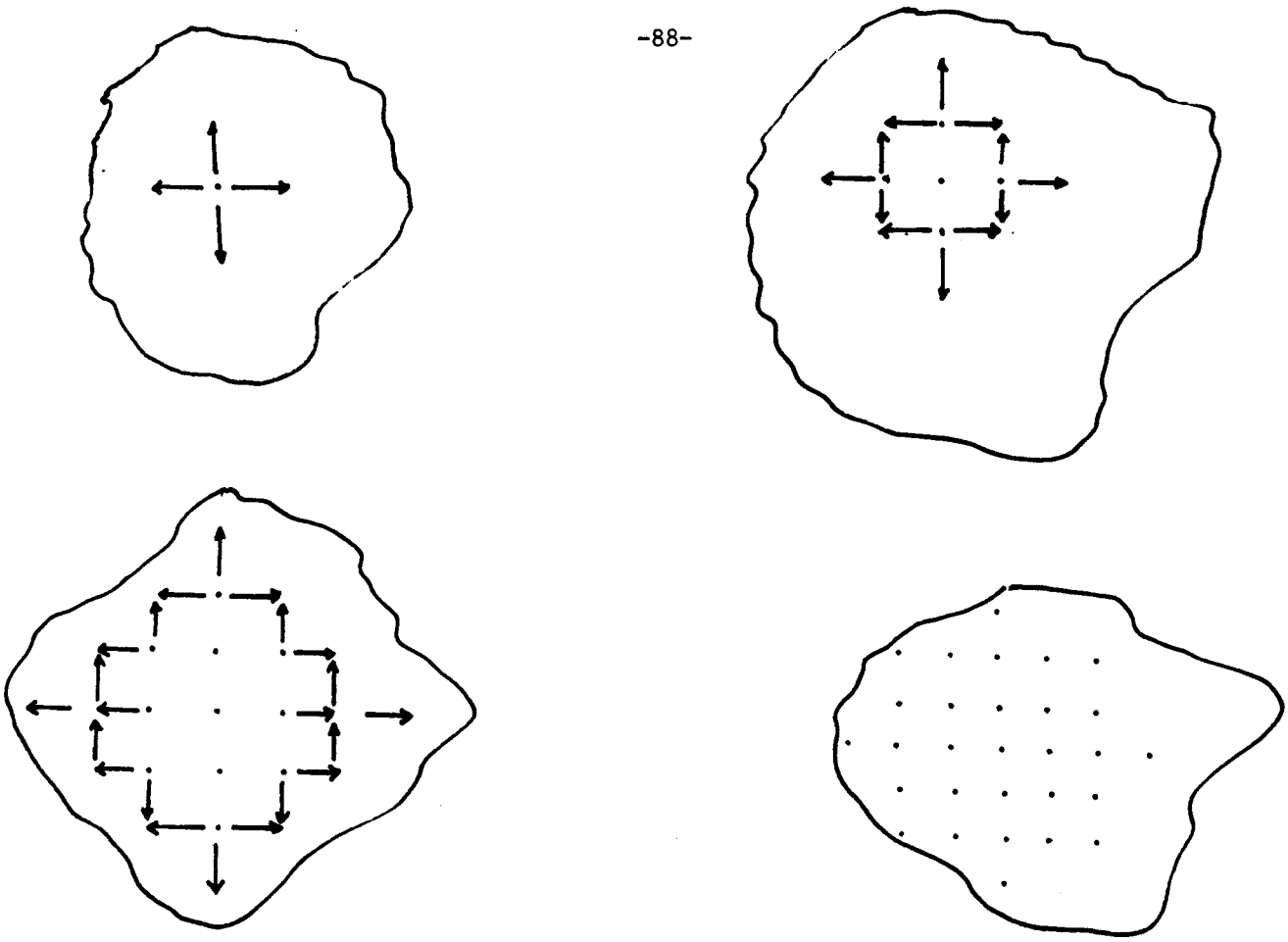


Fig. 4.6 Principe de remplissage

De ce principe, on en déduit l'algorithme récursif suivant :

```
procédure remplir - 4 (x, y, ac, nc)    ; ac ancienne couleur
                                        ; nc nouvelle couleur
        si coul (x,y)= ac  faire
                coul (x,y) = nc
                remplir - 4 (x, y+1, ac, nc)
                remplir - 4 (x+1, y, ac, nc)
                remplir - 4 (x, y-1, ac, nc)
                remplir - 4 (x-1, y, ac, nc)
                ffaire
fremplir-4
```

L'algorithme général, pour visiter tous les pixels d'un contour donné, s'écrirait alors :

```
procédure  remplir - 4 (x, y, contour, nc)
|
|                  si  (x,y) ∉ au contour
|                     et coul (x,y) ≠ nc faire
|                                          |coul (x,y) = nc
|                                          |remplir - 4(x,y+1, nc)
|                                          |remplir - 4(x+1,y, nc)
|                                          |remplir - 4(x, y-1, nc)
|                                          |remplir - 4(x-1, y, nc)
|                                             ffaire
|
|  fremplir
```

On obtient ainsi une procédure simple et hautement récursive facilement exprimable en termes de fonctions LE-LISP et permettant donc d'utiliser les concepts du langage.

4.6.3.2. Algorithme en LE-LISP

La réalisation de l'algorithme de remplissage récursif en LE-LISP a nécessité l'écriture de deux fonctions.

i) une fonction saisie du contour de la tâche à remplir
ii) une fonction récursive de remplissage

1°/ Fonction saisie

Elle permet de :

i) saisir la suite des pixels représentatifs du contour de la tâche
ii) coder un tel contour dans un tableau image de l'écran graphique. Vu la différence des espaces des coordonnées (espace utilisateur travaillant en système de coordonnées normalisées x, y et espace tableau utilisant un système de coordonnées de type tableau classique, nous faisons une transformation simple et triviale des coordonnées utilisateur en coordonnées tableau

iii) saisir le point initial de remplissage (x,y)

iv) d'appeler la fonction remplir avec comme arguments le tableau écran(écran) et le point initial (x,y)

Elle s'écrit en LE-LISP comme suit :

```
(de liste contour (n)

        ; création d'un tableau représentatif de l'écran
        ; exemple d'un écran n x n
        (set écran (makearray n n o ))
        (setq coord (read)) ; saisie des coordonnées à partir d'un
                            ; clavier
        (while coord
                    (aset écran (sub 1(sub n(car(cdr coord))))
                                (car coord) 1)
                    (setq coord (cdr(cdr coord))))

        (setq coord(read)) ; saisie du point initial

        (remplir (sub 1(sub n(car(cdr coord))))
                 (car coord) écran))
```

2°/ Fonction remplir

```
(de remplir (x y écran)
            (cond((<> 1(aref écran x y ))
                 (cond ((<> 2 (aref x y ))
                        (aset écran x y 2)
                        (remplir x (sub1  y )écran)
                        (remplir x (add1  y )écran)
                        (remplir (sub1  x) y écran)
                        (remplir (add1  x) y écran ))))))
```

Pour les besoins de l'expérimentation nous avons écrit une troisème fonction permettant de visualiser sur un écran classique, le résultat du remplissage.

Elle s'écrit en LE-LISP comme suit ;

```
(de impr (écran n)
         (for (x 0 1 n)
              (for (y 0 1 n)
                   (prinflush (aref écran x y )))
                   (terpri 1)))
```

4.6.3.3. Conclusion

L'algorithme décrit ci-dessus est valable quelque soit le polygone (en particulier non convexe ou troué), sous réserve de vérifier la continuité de son contour.

D'autre part son temps d'éxécution peut être estimé comme étant égal à la longueur du chemin conduisant du point de départ au point le plus éloigné dans le polygone [MER 84].

Divers essais ont été faits avec cet algorithme récursif, notamment dans le cas de singularités énoncées par [HEG 85-b].

i) arêtes horizontales ou verticales
ii) sommets
iii) recouvrement d'arêtes

Cet exemple montre comment un des concepts du langage utilisé (en l'occurence la récursivité) permet d'exprimer très facilement certains algorithmes graphiques.

4.6.4. Les arbres quaternaires [FIN 74, BEN 75, KLI 76, DYE 80, JAC 80, SAM 80, BAL 81, MEA 82, OLI 83, CAS 84, OLI 84-a, OLI 84-b, AYA 85, PEL 85]

4.6.4.1. Introduction

Nous avons choisi de développer le modèle des arbres quaternaires car il constitue à notre avis, une bonne synthèse de nos travaux de recherches :

i) sous l'aspect structures de données d'une part, où nous montrerons qu'il existe une forte relation entre la structure de listes au sens LISP, et le modèle de représentation quaternaire.

ii) sous l'aspect algorithmique d'autre part, où nous avons pu exploiter la récursivité du langage LE-LISP pour exprimer certaines fonctions permettant la construction d'arbres quaternaires. Ces fonctions font d'ailleurs partie d'un système fonctionnel basé sur le modèle quaternaire dont nous donnerons quelques spécifications tout au long de ce paragraphe, et que nous nous proposons de réaliser.

4.6.4.2. Présentation

En informatique graphique, la résolution de problèmes complexes liés à une image, tels que le remplissage, l'élimination de surfaces cachées ou la reconnaissance de formes, nécessite une décomposition de l'image de base, en sous-images afin de pouvoir analyser des sous-ensembles simples. L'une des méthodes utilisées, est la décomposition de l'image en quatre sous-images de tailles égales, elles-mêmes décomposées de manière récursive, si la décision à prendre est impossible. L'exemple le plus typique d'une telle approche est l'algorithme de WARNOCK, ou encore l'algorithme de WATKINS [LUC 77, FOL 82].

Cette décomposition permet d'avoir une structure de donnée hiérarchique de l'image.

Ce type de structure de données est de plus en plus utilisé dans le domaine de la conception assistée par ordinateur (C.A.O.), du graphique, du traitement d'images, de la géométrie, etc... Ces structures de données hiérarchiques sont particulièrement adaptées pour stocker et manipuler des objets géométriques et ont été proposées pour la représentation :

i) de points [FIN 74, BIN 75]
ii) de régions planes [SAM 80]
iii) de courbes [SHA 79, BAL 81]

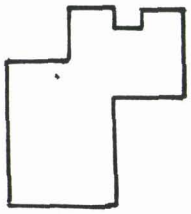
iv) d'objets tridimensionnels [JAC 80, MEA 82, PEL 85]

Nous nous intéresserons à une classe particulière de structures de données hiérarchiques, à savoir les arbres quaternaires.

4.6.4.2.1. La notion d'arbre quaternaire

Elle est basée sur le principe de décomposition récursive de l'image en parties égales (décomposition régulière)

Considérons la région de la figure 4.7.a qui est représentée par un tableau binaire dans la figure 4.7.b (tableau $2^3 \times 2^3$).



(a)

0	0	1	1	0	1	1	0
0	0	1	1	1	1	1	0
1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	0
1	1	1	1	1	0	0	0
1	1	1	1	1	0	0	0
1	1	1	1	1	0	0	0
1	1	1	1	1	0	0	0

(b)

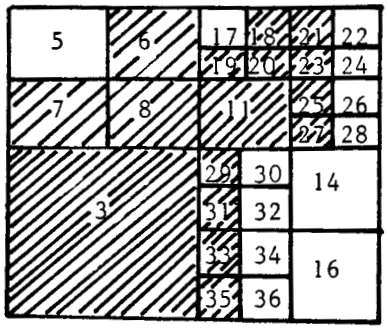
Fig. 4.7 Représentation d'une région

Les 1 correspondent aux pixels qui sont dans la région (internes au contour).

Les 0 correspondent aux pixels qui sont à l'extérieur de la région (externes au contour).

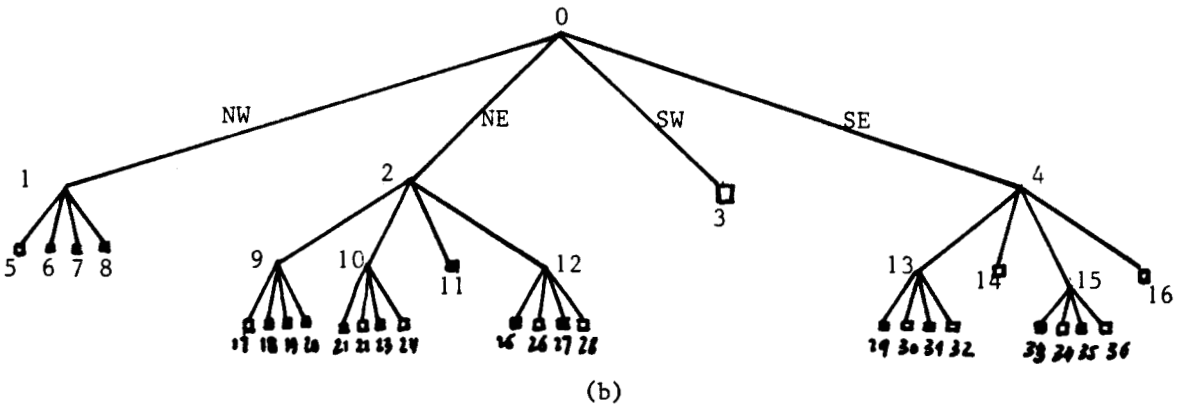
La représentation quaternaire d'une telle image est basée sur des subdivisions successives du tableau en 4 parties égales, la racine de l'arbre représentant toute la région. Si un tableau n'est pas rempli entièrement de 1 ou de 0, il est alors divisé en quadrants, sous-quadrants, etc..., jusqu'à former des blocs homogènes. Eventuellement le processus peut continuer et se terminer à la rencontre de feuilles minimales et indivisibles de l'arbre (à savoir le pixel).

Partant de ce principe, la décomposition de la figure 4.7 nous conduit aux représentations suivantes (Fig. 4.8.a, 4.8.b)



(a)

Fig 4.8 Décomposition maximale de la région en blocs



(b)

Fig. 4.8. Représentation quaternaire de la figure 4.7

Ce processus de décomposition est donc représenté par un arbre de degré 4, c'est-à-dire que chaque noeud a 4 fils. Les 4 fils d'un noeud représentent le quadrant suivant l'ordre NW, NE, SW, SE (Fig. 4.9.a).

NW	NE
SW	SE

Fig. 4.9.a Sens d'orientation des quadrants

On notera qu'il existe différents sens d'orientation d'un quadrant (sens normal ou inverse des aiguilles d'une montre...).
Si un noeud a une taille de $2^m \times 2^m$, ses fils seront de taille $2^{m-1} \times 2^{m-1}$.
De plus les noeuds peuvent être soit de cohérence totale ou homogène (pleins ou vides), soit de cohérence partielle ou hétérogène.

4.6.4.2.2. Avantages du modèle quaternaire

a/ représentation

La décomposition récursive d'une image permet d'obtenir une structure de donnée représentant la scène ou l'image pour différents traitements graphiques (synthèse, traitement, reconnaissance,...)

Dans notre cas, cette représentation est d'autant plus intéressante, puisque nous avons la possibilité, du fait de la structure arborescente, de décrire une scène ou une image par une liste au sens LISP où le jeu des parenthèses ouvrantes et fermantes nous permet de manipuler aisément les noeuds et les feuilles d'un arbre quaternaire.

Exemple :

Soit une décomposition en bloc d'une région donnée, schématisée par le tableau suivant (Fig 4.9-b) :

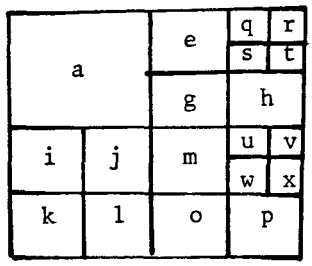


Fig. 4.9.b Décomposition en blocs d'une région donnée

On en déduit la même décomposition hiérarchique sous la forme de liste suivante :

(a b (e f (q r s t)g h) c(i j k l) (d(m n (u v w x) o p)))

Nous utilisons ainsi un des concepts fondamentaux du langage LISP, à savoir la structure de liste. On peut ainsi :

i) obtenir la représentation, sous forme de liste, d'une image quelconque par la méthode de décomposition récursive.

ii) utiliser la structure de liste comme moyen d'entrée, dans un système graphique, d'une scène ou d'une image. La transformation de cette liste sous forme de contour permettra de récupérer l'image ou la scène considérée. Dans un contexte plus général, elle peut être transportée, dans le cas de la norme GKS, vers différentes stations de travail qui peuvent la manipuler, la restituer, la transformer, etc...

iii) une conséquence importante, à notre avis, de cette structure sous forme de liste, est la possibilité de définir sur ce type particulier de liste, un certain nombre d'opérations classiques sur les listes (premier, dernier d'une liste, comparaison de listes, troncature, inverse, etc...).

b / notion de cohérence

Le type de représentation quaternaire conduit à une méthode de traitement qui, contrairement à d'autres qui utilisent la cohérence d'une zone limitée (ligne, colonne), permet de tirer profit de la cohérence des zones variables (quadrants ou partie quaternaire de différents niveaux). On dispose ainsi d'une certaine flexibilité dans le choix de la cohérence pour arrêter la décomposition des différentes branches d'un arbre (même attribut des pixels, liens logiques entre pixels, etc...).

c/ Compactage de l'image

Une conséquence non moins importante des types de représentation et de cohérence décrits plus haut, est la possibilité d'un compactage important lors du stockage d'une image et ceci est un atout non négligeable comparé à la taille mémoire nécessaire pour une image. A titre indicatif, on estime qu'une image sur un écran cathodique couleur actuel représente environ 300K octets d'informations [MER 84].

On obtient les mêmes principes de représentation, de compactage et de cohérence pour des images tri-dimensionnelles [PEL 85]. Dans ce cas, on utilise des arbres à 8 branches appelés octree (arbre octaire).

d/ Manipulation

On peut définir sur les ensembles quaternaires un certain nombre d'opérations du type addition, soustraction, comparaison, extraction, intersection, etc... Il faut néanmoins noter que la description et la manipulation d'une image dans le modèle quaternaire (ou par extension au 3D, octaire), pose un certain nombre de problèmes notamment géométriques. En effet le modèle quaternaire, qui est bi-dimensionnel ne correspond plus à la géométrie classique et il importe d'établir une correspondance entre le modèle cartésien et le modèle quaternaire, d'où la nécessité de concevoir un certain nombre d'algorithmes basés sur ce dernier [PEL 85].

e/ Modélisation

Les arbres quaternaires peuvent être considérés non seulement comme outils de construction d'images ou de scènes, mais également comme outils de modélisation du fait de l'utilisation d'une représentation hiérarchique pour la description d'une image (Cf a/du 4.6.4.2.2).

Le paragraphe suivant permet de mettre en évidence toutes les idées énoncées ici, par la spécification et un essai de réalisation d'un ensemble de fonctions LE-LISP permettant dans une première phase, la construction d'arbres quaternaires à partir d'une image donnée.

4.6.4.3. Algorithme en LE-LISP

4.6.4.3.1. Présentation

Nous nous proposons dans ce paragraphe, de mettre en pratique, à travers les spécifications et l'écriture de fonctions LE-LISP, les idées énoncées tout au long de la présentation des arbres quaternaires.

Les fonctions que nous proposons permettent de construire suivant une approche descendante un arbre quaternaire d'une scène décrite par son contour en entrée.

L'architecture générale de l'algorithme peut être schématisée comme suit (Fig. 4.10) :

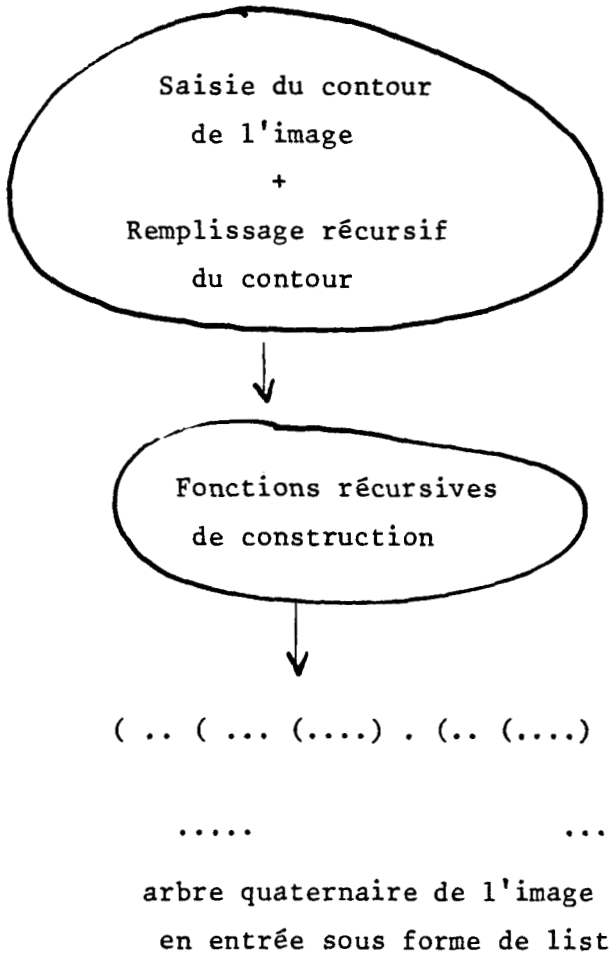


Fig. 4.10 Architecture générale de construction récursive d'un arbre quaternaire.

Donnons maintenant une description détaillée de ces différentes phases :

a/ lère phase : Saisie de remplissage

A partir de la saisie sur clavier des coordonnées du contour d'une image quelconque nous :

i) Construisons un tableau image de l'écran contenant un contour numérisé de l'image.

ii) procédons au remplissage, par une méthode récursive, de l'intérieur du contour.

iii) appelons la fonction récursive de construction de l'arbre quaternaire correspondant.

Ces fonctions sont assez simples et sont inspirées des fonctions que nous avons décrites et réalisées dans l'algorithme récursif de remplissage de tâches (voir leur description et spécification).

b/ 2ème phase : Construction récursive de l'arbre quaternaire

Les différentes fonctions (6 au total) de cette phase réalisent la construction de l'arbre quaternaire associé à une image. Cette construction se fait donc par subdivision successive de l'entrée (image), basée sur le critère d'uniformité d'un quadrant (fonction TYPECRAN).

Ce critère permet de décider si un quadrant est uniforme (homogène) et ne nécessite pas d'autres subdivisions, ou s'il doit être subdivisé en 4 sous-quadrants. Si le quadrant analysé est uniforme il est stocké dans l'arborescence avec le type homogène par la fonction INSERT-FEUILLE. Dans le cas contraire, on stocke un noeud de type hétérogène dans l'arborescence avec la fonction INSERT-NOEUD, et on appelle la fonction ARQUATER pour réaliser la subdivision et l'analyse de ce quadrant.

4.6.4.3.2. Fonctions LE-LISP

a/ Fonction ARQUATER

Fonction de base de l'algorithme, elle est récursive et permet la construction de l'arbre quaternaire (Fig. 4.11)

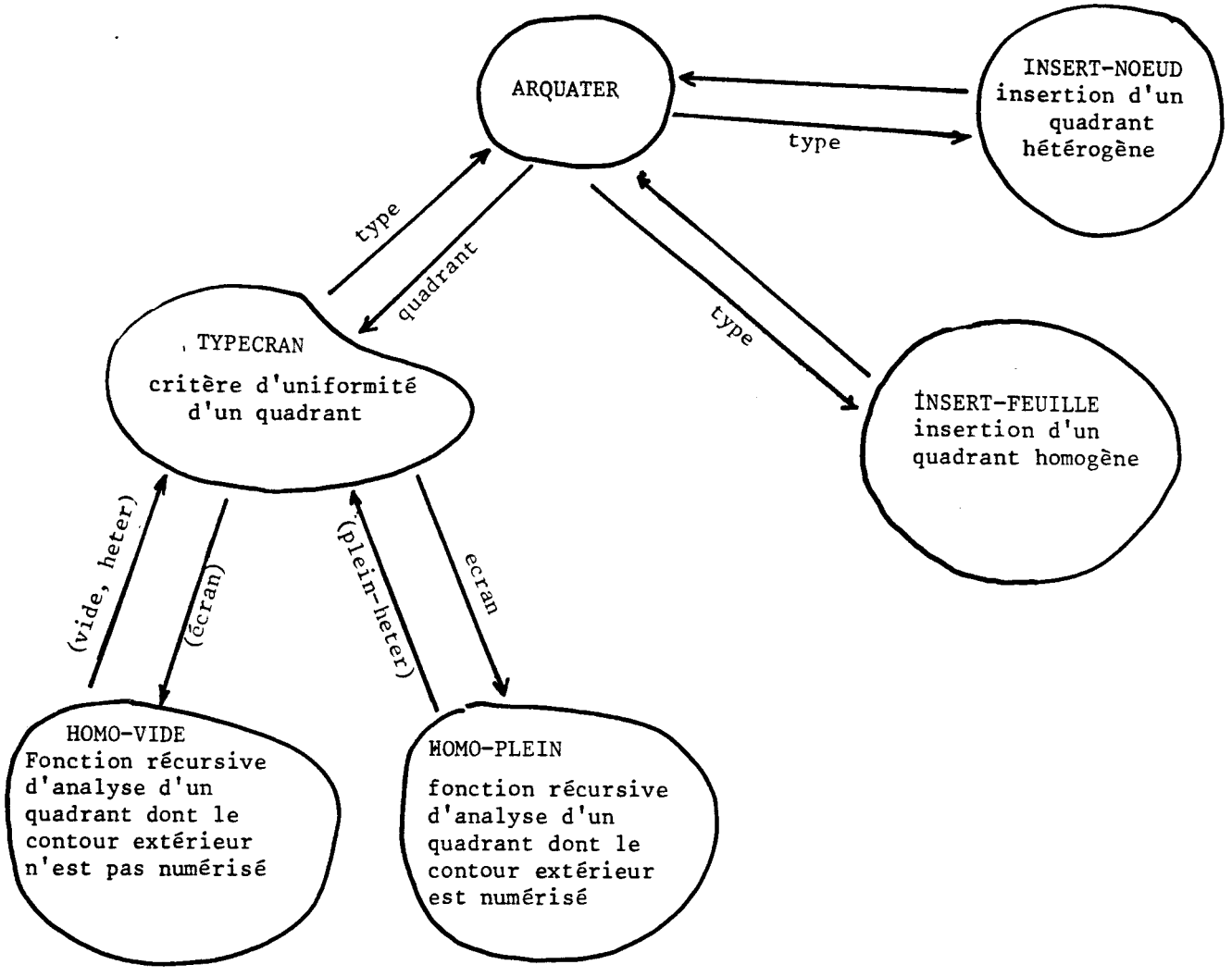


Fig. 4.11 Architecture des fonctions de construction

Elle fait appel pour les besoins de la construction à différentes fonctions (voir figure 4.11) que nous spécifions ci-dessous.

b/ Fonction TYPECRAN

Elle permet d'analyser le critère d'uniformité d'un quadrant. Cette uniformité peut être de 3 types selon le schéma ci-après (Fig. 4.12)

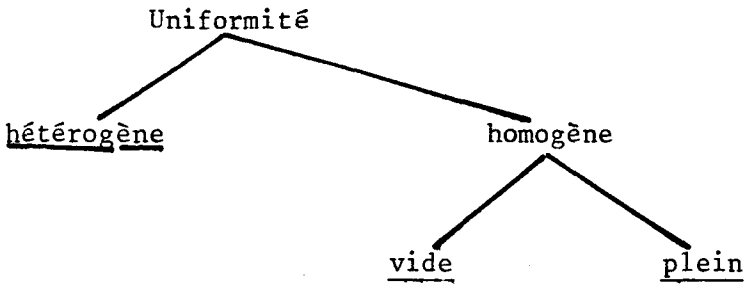


Fig. 4.12 Caractéristiques d'un quadrant

Elle procède de la manière quivante pour déterminer l'un des 3 critères :

* Partant d'un quadrant donné, on construit les vecteurs lignes et vecteurs colonnes de ses 4 extrêmités. Ces 4 vecteurs sont comparés, en terme de valeurs, avec deux vecteurs de base (vect-plein et vect-vide) et peuvent donner les résultats suivants (Fig. 4.13) :

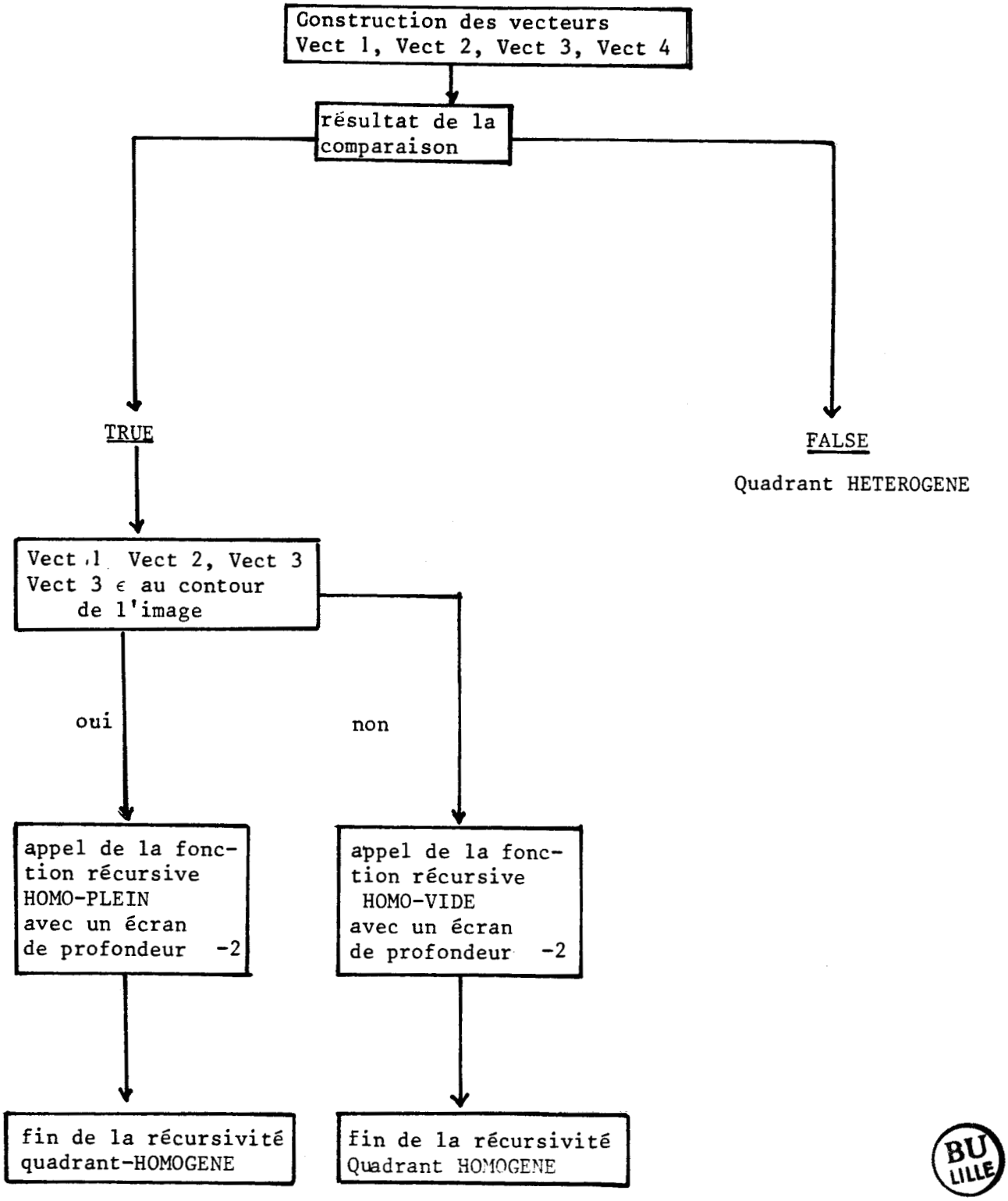


Fig. 4.13 Méthode de détermination de l'uniformité d'un quadrant.

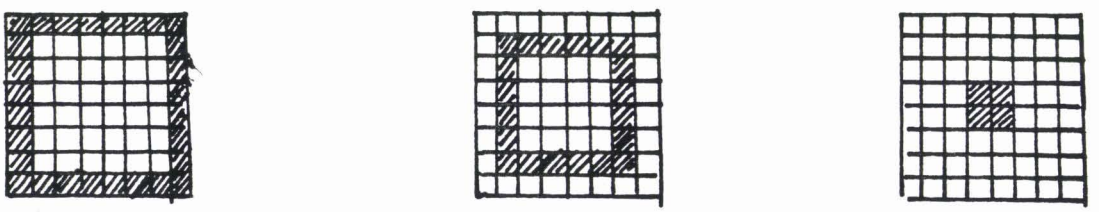
Expliquons cette méthode :

Soit un quadrant Qi de coordonnées origines (x_o y_o) et de taille (long), et soient les vecteurs V1, V2, V3, V4 représentatifs de ses lignes et de ses colonnes les plus extérieures.

CAS 1 : La comparaison des valeurs des vecteurs avec les vecteurs de base (vect-plein et vect-vide) donne un résultat faux. On en déduit que le quadrant est hétérogène puisqu'il y a présence simultanée de 1 et de 0 dans les pixels de son contour le plus externe.

CAS 2 : La comparaison avec le vecteur (vect-plein), qui est un vecteur dont l'ensemble des éléments est égal à 1, donne un résultat TRUE. On appelle une fonction récursive d'analyse par profondeur descendante qui retournera comme valeur le type du quadrant analysé (Fig. 4.14)

CAS 3 : La comparaison avec le vecteur (vect-vide), qui est un vecteur dont l'ensemble des éléments est égal à 0, donne un résultat TRUE. On appelle alors une fonction récursive d'analyse par profondeur récursive comme dans le cas 2 (Fig. 4.14).



a) Profondeur maximale i

b) Profondeur i-2

c) Profondeur minimale

Fig. 4.14 Analyse par profondeur descendante d'un quadrant

c) Fonction HOMO-VIDE

C'est une fonction récursive d'analyse par profondeur descendante d'un quadrant dont le contour extérieur est égal au vecteur de base (vect-vide). A chaque pas de récursivité on construit dynamiquement le vecteur de base (vect-vide) en fonction de la profondeur. (Fig. 4.14).

Cette analyse récursive continue tant que les 4 vecteurs V1, V2, V3, V4 sont égaux au vecteur de base vect-vide. Si elle atteint le centre du quadrant on en déduit que l'uniformité de ce quadrant est de type homogène-vide. Dans le cas contraire on a à faire à un quadrant de type hétérogène.

d)Fonction HOMO-PLEIN

Elle est identique à la fonction précédente sauf qu'elle utilise un vecteur de base différent (vect-plein). Si l'analyse aboutit au centre du quadrant, celui-ci est de type homogène-plein. Dans le cas contraire il est de type hétérogène. Dans le cas de cette fonction, nous avons analysé le quadrant jusqu'à son centre pour tenir compte des images trouées. Dans le cas où l'on aurait à faire à des figures non trouées, cette analyse serait superflue et il suffirait de s'arrêter à l'analyse du contour le plus externe d'un quadrant.

e) Fonction INSERT-NOEUD

Elle est mise en oeuvre, lors de la détection d'un noeud hétérogène dont il faut réaliser la subdivision. Dans cette première version elle permet d'insérer le type du noeud dans l'arbre quaternaire de l'image.

f) Fonction INSERT-FEUILLE

Deuxième fonction de construction de l'arbre quaternaire, elle est appelée lorsque l'on rencontre un quadrant qui ne nécessite plus de divisions (quadrant homogène ou indivisible). Elle insère, dans sa première version, le type de la feuille dans l'arbre quaternaire de l'image.

4.6.4.4. Conclusion

A travers cette première tentative d'écriture, de fonctions LE-LISP pour l'utilisation des arbres quaternaires, tentative qui demande bien sûr un travail complémentaire, nous avons voulu mettre en évidence l'utilisation des deux notions fondamentales de LISP, la structure de liste et la récursivité :

i) la structure de liste s'est avérée un bon outil pour représenter des objets images, et sa combinaison avec le modèle quaternaire permet à la fois d'exprimer et d'exploiter la structure d'une image.

ii) la récursivité qui est représentée dans toute sa puissance avec le modèle quaternaire. Cet exemple montre bien quand l'esprit d'un modèle ou d'une technique graphique s'y prête, la récursivité peut être très avantageuse pour écrire des fonctions en LISP.

Comme pour les autres algorithmes détaillés précédemment, deux problèmes subsistent encore et pour lesquelles une reconsidération des environnements de programmation LISP est à envisager :

i) Une réallocation de l'espace mémoire aux différents objets du langage en tenant compte du type d'objet graphique et de sa complexité.

ii) une redéfinition de la pile d'évaluation de l'interpréteur qui s'est avérée tout au long de notre expérimentation, très réduite pour supporter un taux de récursivité élevé.

4.7. Conclusion

En guise de conclusion sur cette partie expérimentation, nous nous proposons de voir si nous avons pu apporter des éléments de réponse à la question que nous nous étions posé au début de nos travaux et relative à l'éventuel apport des langages applicatifs au graphique.

Au vu de cette première expérimentation, il nous semble que l'apport de langages tels que LISP se situe à trois niveaux :

i) niveau structure de données où la notion de liste s'est avérée très bien adaptée pour décrire des images. Elle joue ainsi le rôle de fichier intermédiaire structuré (Cf. Chap. sur les relations entre structures de données et systèmes graphiques) dans un système graphique interactif. La composante d'un tel fichier peut être une suite de fonctions dont la composition permet de donner une description de l'image.

La notion de liste constitue également, par son aspect hiérarchique, un modèle intéressant pour la description de scènes, pour lequel un certain nombre de fonctions de composition et de manipulation restent à définir.

ii) niveau algorithmique où la récursivité permet d'exprimer de façon naturelle (syntaxe mise à part !) un certain nombre d'algorithmes graphiques. A ce niveau, il nous semble qu'une distinction doit être faite entre les algorithmes intuitivement récursifs et ceux nécessitant une transformation. Les premiers s'expriment très facilement (remplissage par exemple) alors que les seconds nécessitent une transformation qui n'est pas toujours évidente (tracé de segment par exemple). Cette transformation peut dans certains cas remettre en cause leur efficacité par rapport à d'autres types d'algorithmes (séquentiels ou parallèles par exemple).

iii) niveau méthode de construction de logiciels par la décomposition d'un algorithme en une série de fonctions LISP (programmation de type modulaire).

Il faut souligner également que notre expérimentation doit être beaucoup plus développée pour apporter des éléments de réponse notamment en ce qui concerne la complexité des algorithmes en espace mémoire et en temps de calcul pour pouvoir juger de leur réelle efficacité par rapport à des algorithmes parallèles, voire même séquentiels.

D'autre part, notre expérimentation a montré certaines limites du système LE-LISP sous MULTICS que nous avons utilisé. On est arrivé en effet très vite à saturer le système notamment au niveau de la gestion du tas (pile de l'interpréteur) avec l'éxécution des fonctions tracé de segment et remplissage notamment. On s'est rendu compte que le système ne pouvait pas admettre des écrans image sous forme de listes de plus de 140x140. D'autre part un taux de récursivité élevé saturait très vite le système (remplissage imposssible d'un carré supérieur à 32x32).

Ceci provient à notre avis du fait que les implémentations de LISP, ne sont pas adaptées à traiter des problèmes graphiques nécessitant un espace mémoire important et un niveau de récursivité très élevé. Il y a donc une certaine adaptabilité à réaliser entre les environnements LISP et les problèmes graphiques. Comme inconvénients des systèmes LISP on peut en citer quelques uns que nous avons rencontré lors de notre expérimentation :

i) systèmes assez lourds et gros consommateurs de mémoire

ii) lisibilité très réduite des programmes

CONCLUSION GENERALE

Nous voudrions, avant de présenter nos conclusions, rappeler les motivations qui sont à l'origine de nos travaux.

La synthèse d'images, de part la masse d'informations (souvent complexes) à traiter pour la génération de pixels composants les images, et la nécessité de disposer de temps très brefs pour permettre une bonne interactivité, fait partie des applications informatiques pour lesquelles on essaie de mettre en oeuvre des architectures et des logiciels répondant, le mieux possible, à ces deux principales contraintes qui sont l'espace mémoire et le temps de calcul.

Devant cette situation, et après avoir analysé les besoins du graphique en matière de structures de données et d'algorithmique, notre travail est une approche par l'utilisation de la structure de liste pour représenter des images, et la récursivité pour exprimer des algorithmes graphiques. Cette longue étude, étant donné le nombre de fonctions à développer, n'est ici qu'entamée et nécessite donc un travail complémentaire.

Notre tentative d'utilisation des notions de listes et de récursivité pour traiter certains problèmes graphiques, peut contribuer à améliorer leurs résolutions et initialiser des études dans la conception de modèles de représentation d'images à base de listes, ainsi que la conception d'une algorithmique graphique récursive. Ceci pose le problème, que nous avons abordé dans la partie réalisation (Cf 4.7) de la redéfinition des systèmes LISP pour les adapter aux problèmes graphiques (niveau élevé de récursivité, tailles importantes de listes).

La conception d'algorithmes, apparaît donc comme un prolongement logique de nos travaux actuels. Il est à noter que la structure de liste pour représenter des images et les avantages qu'elle procure, met à jour de nouveaux axes de recherches dans ce domaine. Ainsi l'aspect hiérarchique d'une image, que nous avons essayé de mettre en évidence, à travers les arbres quaternaires, peut-être conservé et utilisé judicieusement pour développer un système fonctionnel de synthèse d'images à base d'arbres quaternaires avec utilisation des listes.

En ce qui concerne les développements directs de nos travaux, nous citerons quelques uns qui sont d'ores et déjà à l'étude ou en voie de l'être,

Un des premiers développement consisterait à partir de listes d'images, d'imaginer un ensemble de fonctions sur ces listes, tels que CAR, CDR, EQUAL, CONS, LAST, FIRST, ADD, etc.. On disposerait ainsi d'un système extensible pour la manipulation d'images à travers leurs listes,

Un deuxième développement aurait pour objectif d'étendre la notion d'arbres quaternaires, pour traiter des objets tridimensionnels. Cette extension, aux arbres octaires, permettrait d'aborder avec un langage tel que LISP et la structure de liste, des problèmes de type surfaces cachées, reconnaissance de forme, découpage, etc ..., en travaillant sur des représentations octaires des images (comparaison de listes d'images).

D'autres retombées directes de nos travaux nous paraissent devoir susciter, à moyen ou plus long terme, de nouveaux thèmes de recherche.

C'est le cas notamment de l'aspect modélisation des images, à l'aide de la structure de liste. Notre approche ayant été concentrée sur l'objet pixel, nous proposons d'utiliser la notion de liste pour aborder le problème de modélisation des objets graphiques à un niveau macro-pixel. Une première esquisse de cette approche a déjà été donnée avec les arbres quaternaires (construction d'une image à partir de son arbre quaternaire). On pourrait ainsi décrire naturellement sous forme de listes hiérarchiques (listes multiniveaux) une image à base d'objets prédéfinis.

Cet aspect modélisation nous amène à parler d'un autre thème de recherche qui nous semble très prometteur et très enrichissant, à savoir une approche langages orientés objets.

IL s'agit ici d'utiliser de tels langages pour l'expression d'un certain nombre de problèmes graphiques. De manière tout à fait intuitive cette approche semble très interessante car ce type de langage (LOGO et plus particulièrement SMALLTALK) dispose d'un certain nombre de concepts graphiques. Ainsi SMALLTALK-80 dispose d'un noyau graphique et d'une classe PEN permettant de réaliser un grand nombre de primitives graphiques.

Un dernier axe de recherche orienté normalisation, consisterait à envisager une synthèse LISP-GKS et permettrait d'aborder les problèmes d'interface entre le langage LISP et la norme GKS.

Nous espérons que nos travaux, auront prouvé que les recherches en matière de représentation des images et de conception d'algorithmes graphiques sont loin d'être terminées et que ces problèmes peuvent être abordés selon différentes approches, malgré la qualité fort satisfaisante des images produites actuellement.

ANNEXE

FONCTIONS DE
TRACE DE SEGMENT.

```
;                F O N C T I O N  APTRASEG


(de aptraseg (n)


  ; aptraseg fait partie d'un ensemble de fonctions realisant un
  ; trace de segment par methode dichotomique. Elle permet :
  ; 1/ d'initialiser un tableau representatif de l'ecran
  ; 2/ de saisir les coordonnees du segment a tracer
  ; 3/ de faire une transformation des coordonnees espace utili-
  ;    sateur en coordonnees tableau
  ; 4/ d'afficher les extremites du segment
  ; 5/ d'appeler la fonction recursive de trace 'traseg'
  ; Argument : taille de l'espace ecran (nombre d'elements)

                  (setq liseg ())
                  (setq coord (read))
                  (setq ecran (makearray n n 0))
                  (newr liseg (differ (differ n (nth 1 coord)) 1.))
                  (newr liseg (nth 0 coord))
                  (newr liseg (differ (differ n (nth 3 coord)) 1.))
                  (newr liseg (nth 2 coord))
                  (aset ecran (fix (car liseg)) (fix (nth 1 liseg)) 1 )
                  (aset ecran (fix (nth 2 liseg)) (fix (nth 3 liseg)) 1 )
                  (traseg liseg ecran))
```

```
;                F O N C T I O N  TRASEG


(de traseg (liseg ecran)

 ; Fonction recursive de calcul et d'affichage des points d'un segment
 ; par la methode dichotomique. Appelee par la fonction 'aptraseg' elle
 ; appelle deux autres fonctions :
 ; 1/ la fonction 'parent' qui retourne les parties entieres des
 ;    coordonnees d'un point.
 ; 2/ la fonction 'genseg' qui permet de generer a partir d'un point
 ;    milieu, les sous-segments correspondants.
 ; Elle utilise comme structures de donnees 3 listes et un tableau :
 ; - Liseg  : liste des sous-segments a traiter
 ; - Coordent : liste des coordonnees entieres du segment en cours
 ; - Ecran  : espace ecran pour la generation des points du segment
 ; A noter que les calculs se font en reel.

          (setq coordent (parent (firstn 4 liseg)))
          (cond ((<> (nth 0 coordent) (nth 2 coordent))
                 (cond ((<> (nth 1 coordent) (nth 3 coordent))
                        (aset ecran (fix (nth 6 coordent))
                                    (fix (nth 7 coordent)) 1)
                        (setq liseg (genseg liseg (nth 4 coordent)
                                                  (nth 5 coordent)))
                        (traseg liseg ecran))))))
```

```
;                F O N C T I O N  GENSEG


(de genseg (liseg xm ym)

 ; La fonction 'genseg' permet de generer a partir :
 ; - des coordonnees debut et fin d'un segment courant
 ; - et de ses coordonnees milieu
 ; - les deux sous-segments resultant de la division par 2.
 ; Arguments :
 ; - liste des sous-segments a subdiviser (liseg)
 ; - (xm ym) coordonnees milieu du sous-segment courant

                (newr liseg (nth 0 liseg))
                (newr liseg (nth 1 liseg))
                (newr liseg xm)
                (newr liseg ym)
                (newr liseg xm)
                (newr liseg ym)
                (newr liseg (nth 2 liseg))
                (newr liseg (nth 3 liseg))
                (setq liseg (nthcdr 4 liseg)))
```

```
;                F O N C T I O N  PARENT


(de parent (coordent)

 ; La fonction 'parent' permet de calculer les parties entieres
 ; des coordonnees d'un point. Elle utilise une liste (coordent)
 ; contenant :
 ; - les coordonnees debut et fin du segment en cours
 ; - les coordonnees du milieu du segment en cours
 ; Argument :
 ; - coordonnees reelles du segment courant (coordent)


              (cond ((> (nth 0 coordent) (fix (nth 0 coordent)))
                     (newr coordent (plus (fix (nth 0 coordent)) 1.)))
                    (T(newr coordent (nth 0 coordent))))
              (cond ((> (nth 1 coordent) (fix (nth 1 coordent)))
                     (newr coordent (plus (fix (nth 1 coordent)) 1.)))
                    (T(newr coordent (nth 1 coordent))))
              (cond ((> (nth 2 coordent) (fix (nth 2 coordent)))
                     (newr coordent (plus (fix (nth 2 coordent)) 1.)))
                    (T(newr coordent (nth 2 coordent))))
              (cond ((> (nth 3 coordent) (fix (nth 3 coordent)))
                     (newr coordent (plus (fix (nth 3 coordent)) 1.)))
                    (T(newr coordent (nth 3 coordent))))
              (newr coordent (divide(plus(car coordent)(nth 2 coordent))2.))
              (newr coordent (divide (plus (nth 1 coordent)
                                           (nth 3 coordent)) 2.))
              (cond((> (nth 8 coordent) (fix (nth 8 coordent)))
                        (newr coordent (plus (fix (nth 8 coordent)) 1.)))
                   (T(newr coordent (nth 8 coordent))))
              (cond((> (nth 9 coordent)(fix(nth 9 coordent)))
                        (newr coordent (plus (fix (nth 9 coordent)) 1.)))
                   (T(newr coordent (nth 9 coordent))))
              (setq coordent (nthcdr 4 coordent)))
```

FONCTIONS DE REMPLISSAGE

```
;                    F O N C T I O N  LISTCONTOUR


(de listcontour (n)

 ; La fonction listcontour permet :
 ; 1/ de saisir un contour d' une tache (suite de nombres entiers)
 ; 2/ de faire une transformation des coordonnees de l' espace
 ; utilisateur en coordonnees de l' espace tableau (les indices
 ; de tableau en Lelisp commencent par 0)
 ; 3/ de numeriser le contour dans le tableau ecran
 ; 4/ de saisir le point de depart pour le remplissage
 ; 5/ d' appeler la fonction remplir

                  (setq coord (read))
                  (setq ecran (makearray n n 0))
                  (while coord
                         (aset ecran (sub1 (sub n (car (cdr coord))))
                                     (car coord) 1)
                         (setq coord (cdr (cdr coord))))
                         (setq coord(read))
                         (remplir (sub1 (sub n (car (cdr coord))))
                                  (car coord) ecran))
```

```
;                F O N C T I O N  REMPLIR


(de remplir (x y ecran)

  ; Fonction recursive permettant de remplir une tache quelconque dans
  ; un espace ecran suivant le principe du jeu de la vie. Le contour
  ; de la tache est materialise par le chiffre 1, la notion de couleur
  ; par le chiffre 2. L'extension a un attribut couleur est relative -
  ; ment facile.
  ; Arguments de la fonction :
  ; Ecran (espace ecran)
  ; x y representent les coordonnees du point de depart pour
  ; le remplissage. Cette fonction est appelee par une autre fonction
  ; 'listcontour' (voir sa description)

            (cond ((<> 1 (aref ecran x y))
                   (cond ((<> 2 (aref ecran x y))
                          (aset ecran x y 2)
                          (remplir x (sub1 y) ecran)
                          (remplir x (add1 y) ecran)
                          (remplir (sub1 x) y ecran)
                          (remplir (add1 x) y ecran))))))
```

```
;                    F O N C T I O N  IMPR


(de impr (ecran n)

 ; La fonction impr permet de visualiser sur ecran normal
 ; un tableau representatif d' un ecran image (ici n x n)

            (for (x 0 1 (sub1 n))
                 (for (y 0 1 (sub1 n))
                      (prinflush (aref ecran x y)))
                      (terpri 1)))
```

FONCTIONS CONSTRUCTION D'ARBRES QUATERNAIRES

```
;                    F O N C T I O N  ARQUATER


(de arquater (ecran x0 y0 long arbre)

 ; Fonction recursive de base de l'algorithme de construction d'arbres
 ; quaternaires.
 ; Arguments :
 ; - tableau ecran
 ; - coordonnees origines de l'ecran (x0 y0)
 ; - longueur du quadrant (long)
 ; - liste de l'arborescence (arbre)
 ; Elle fait appel a d'autres fonctions pour la construction (TYPECRAN,
 ; INSERT-NOEUD, INSERT-FEUILLE)
 ; Elle travaille sur la base du critere d'uniformite d'un quadrant pour
 ; decider de sa subdivision ou non (voir la fonction TYPECRAN)

      (cond ((> long 0)
             (typecran ecran x0 y0 long)
             (cond ((equal type "heter")
                    (setq arbre (insert-noeud arbre))
                    (arquater ecran (differ x0 (div long 2)) y0
                                    (div long 2) arbre)
                    (arquater ecran (differ x0 (div long 2))
                                    (plus y0 (div long 2))
                                    (div long 2) arbre)
                    (arquater ecran x0 y0 (div long 2) arbre)
                    (arquater ecran x0 (plus y0 (div long 2))
                                       (div long 2) arbre))
                   (T(setq arbre (insert-feuille type arbre)))))
             (T(setq arbre arbre))))
```

```
;                    F O N C T I O N  MODQUATER


(de modquater (n)

 ; MODQUATER est la premiere fonction d'un ensemble
 ; permettant la construction d'arbres quaternaires a partir
 ; d'une image. Elle realise les actions suivantes :
 ; 1/ saisie du contour
 ; 2/ creation du tableau image de l'ecran
 ; 3/ saisie du point de depart pour le remplissage de l'image
 ; 4/ appel de la fonction de remplissage (remplir-quater)
 ; 5/ appel de la fonction de construction ARQUATER
 ; Argument : n (taille de l'ecran)

              (setq ecran (makearray n n 0))
              (setq coord (read))
              (while coord
                     (aset ecran (sub1 (sub n (car (cdr coord))))
                                 (car coord) 2)
              (setq coord (cdr(cdr coord))))
              (setq coord (read))
              (remplir-quater (sub1 (sub n (car (cdr coord))))
                              (car coord) ecran n)
              (setq arbre (arquater ecran (sub1 n) 0 n NIL)))
```

```
;                F O N C T I O N  REMPLIR-QUATER


(de remplir-quater (x y ecran n)

 ; Fonction recursive de remplissage integree au systeme de fonctions
 ; realisant la construction d'arbres quaternaires. Elle permet de
 ; remplir les taches de l'image avec la valeur 1, l'exterieur est
 ; materialise par le chiffre 0.
 ; Arguments :
 ; - ecran (espace ecran)
 ; - x y (coordonnees du point origine pour le remplissage)
 ; - n (nombre de pixels de l'ecran)

   (cond ((<> 2 (aref ecran x y))
          (cond ((<> 1 (aref ecran x y))
                 (aset ecran x y 1)
                 (remplir-quater x (sub1 y) ecran n)
                 (remplir-quater x (add1 y) ecran n)
                 (remplir-quater (sub1 x) y ecran n)
                 (remplir-quater (add1 x) y ecran n)))))
   (for (x0 0 1 (sub1 n))
        (for (y0 0 1 (sub1 n))
             (cond ((= (aref ecran x0 y0) 2)
                    (aset ecran x0 y0 1))))))
```

```
;                F O N C T I O N  TYPECRAN


(de typecran (ecran x0 y0 taille)

 ; Fonction d'analyse du critere d'uniformite d'un quadrant. Elle
 ; utilise la notion de profondeur descendante d'un quadrant pour
 ; determiner son type. Elle retourne comme valeur les 3 types suivants :
 ; - HETER = quadrant heterogene a subdiviser
 ; - PLEIN = quadrant homogene plein (interieur au contour de l'image)
 ; - VIDE  = quadrant homogene vide (exterieur au contour de l'image)
 ; Elle fait appel a la fonction 'fnoyau' pour determiner le type
 ; du quadrant.

      (setqq vect1 () vect2 nil vect3 () vect4 nil)
      (setq vect-plein (makelist taille 1))
      (setq vect-vide  (makelist taille 0))
      (cond((= taille 1)
            (cond ((equal (aref ecran x0 y0) 1)
                   (setq type "plein"))
                  (T(setq type "vide"))))
           (T(setq type (fnoyau x0 y0 taille ecran vect-plein vect-vide)))))
```

```
;                    F O N C T I O N  FNOYAU


(de fnoyau (x0 y0 taille ecran vect-plein vect-vide)

 ; La fonction 'fnoyau' permet de construire les 4 vecteurs
 ; lignes et colonnes les plus externes d'un quadrant i. Ces vecteurs
 ; sont compares a deux vecteurs de base (vect-plein et vect-vide) et
 ; peuvent donner lieu a 3 situations :
 ; a) vecteurs differents ----> quadrant heterogene
 ; b) vecteurs egaux a vect-plein ----> appel fonction 'homo-plein'
 ; c) vecteurs egaux a vect-vide  ----> appel fonction 'homo-vide'


        (for (i x0 -1 (plus (differ x0 taille) 1))
             (newr vect1 (aref ecran i y0))
             (newr vect2 (aref ecran i (differ (plus y0 taille) 1))))
        (for (i y0  1 (differ (plus y0 taille) 1))
             (newr vect3 (aref ecran x0 i))
             (newr vect4 (aref ecran (plus (differ x0 taille) 1) i)))
        (setq type-plein (and (equal vect1 vect-plein)
                              (equal vect2 vect-plein)
                              (equal vect3 vect-plein)
                              (equal vect4 vect-plein)))
        (setq type-vide (and (equal vect1 vect-vide)
                             (equal vect2 vect-vide)
                             (equal vect3 vect-vide)
                             (equal vect4 vect-vide)))
        (cond ((equal type-plein T)
               (setq type (homo-plein ecran x0 y0 taille)))
              ((equal type-vide T)
               (setq type (homo-vide ecran x0 y0 taille)))
              (T(setq type "heter"))))
```

```
;                F O N C T I O N  HOMO-PLEIN


(de homo-plein (ecran xa ya size-cour)

 ; Fonction recursive d'analyse par profondeur descendante d'un ecran
 ; dont le contour exterieur est egal a 1.
 ; Si l'ensemble du quadrant est a 1 elle retourne le type PLEIN
 ; dans le cas contraire elle retourne type HETER.

      (setqq vect1 () vect2 () vect3 () vect4 ())
      (setq size-cour (sub size-cour 2))
      (cond((> size-cour 0)
            (setq xa (sub1 xa))
            (setq ya (add1 ya))
            (setq vect-plein (makelist size-cour 1))
            (for (i xa -1 (plus (differ xa size-cour)1))
                 (newr vect1 (aref ecran i ya))
                 (newr vect2 (aref ecran i (differ (plus ya size-cour) 1))))
            (for (i ya 1 (differ (plus ya size-cour)1))
                 (newr vect3 (aref ecran xa i))
                 (newr vect4 (aref ecran (plus (differ xa size-cour) 1) i)))
            (cond ((and (equal vect1 vect-plein)
                        (equal vect2 vect-plein)
                        (equal vect3 vect-plein)
                        (equal vect4 vect-plein))
                        (homo-plein ecran xa ya size-cour))
                  (t(setq type "heter"))))
           (T(setq type "plein"))))
```

```
;                F O N C T I O N  HOMO-VIDE


(de homo-vide (ecran xa ya size-cour)

 ; Fonction recursive d'analyse par profondeur descendante d'un ecran
 ; dont le contour exterieur est egal a 0.
 ; Si l'ensemble du quadrant est a 0 elle retourne le type VIDE
 ; dans le cas contraire elle retourne type HETER.

     (setqq vect1 () vect2 nil vect3 () vect4 nil)
     (setq size-cour (sub size-cour 2))
     (cond((> size-cour 0)
           (setq xa (sub1 xa))
           (setq ya (add1 ya))
           (setq vect-vide (makelist size-cour 0))
           (for (i xa -1 (plus(differ xa size-cour)1))
                (newr vect1 (aref ecran i ya))
                (newr vect2 (aref ecran i (differ (plus ya size-cour) 1))))
           (for (i ya  1 (differ (plus ya size-cour) 1))
                (newr vect3 (aref ecran xa i))
                (newr vect4 (aref ecran (plus (differ xa size-cour) 1) i)))
           (cond ((and (equal vect1 vect-vide)
                       (equal vect2 vect-vide)
                       (equal vect3 vect-vide)
                       (equal vect4 vect-vide))
                  (homo-vide ecran xa ya size-cour))
                 (t(setq type "heter"))))
          (T(setq type "vide"))))
```

```
;                    F O N C T I O N  INSERT-NOEUD


(de insert-noeud (arbre)

 ; Cette fonction permet d'inserer dans la liste representant
 ; l'arborescence d'un ecran, un quadrant heterogene. Pour un
 ; premier essai on insere la lettre N pour signaler que c'est
 ; un noeud qui aura donc 4 fils.

        (newr arbre "N"))
```

```
;                F O N C T I O N  INSERT-FEUILLE


(de insert-feuille (type arbre)

 ; Cette fonction permet d'inserer dans la liste representant
 ; l'arborescence d'un ecran, une feuille (c'est a dire un
 ; quadrant indivisible) avec son type. Pour un premier essai
 ; on insere la lettre V quand on a un quadrant de type vide
 ; ou la lettre P quand on a un quadrant plein.


      (cond ((equal type "vide")
             (newr arbre "V"))
            (T(newr arbre "P"))))
```

```
;                F O N C T I O N  QUAD-TREE


(de quad-tree (arbre tree niveau)

 ; La fonction 'quad-tree' permet de transformer l'arbre lineaire
 ; d'une image produit par la fonction 'arquater' en un arbre
 ; hierarchise de type quaternaire.
 ; Arguments :
 ; - arbre   ------> liste contenant l'arbre lineaire
 ; - tree    ------> liste de l'arbre hierarchise initialisee a nil
 ; - niveau  ------> liste de comptage des niveaux de l'arborescence


   (cond((equal (length arbre) 0)
         (newr tree ")")
         (print tree))
        (T(cond((equal (car arbre) 'N)
                (cond((eq (nth (sub1 (length niveau)) niveau) 4)
                      (setq niveau (firstn (sub1 (length niveau))
                                           niveau))
                      (setq nc (last niveau))
                      (setq nc (add1 (nth 0 nc)))
                      (setq niveau (firstn (sub1 (length niveau))
                                           niveau))
                      (newr niveau nc)
                      (newr tree ")")
                      (newr tree (car arbre))
                      (newr niveau 0)
                      (newr tree "(")
                      (quad-tree (cdr arbre) tree niveau))
                     (T(newr niveau 0)
                       (newr tree (car arbre))
                       (newr tree "(")
                       (quad-tree (cdr arbre) tree niveau))))
               (T(cond ((eq (nth (sub1 (length niveau)) niveau) 4)
                        (setq niveau (firstn (sub1 (length niveau))
                                             niveau))
                        (setq nc (last niveau))
                        (setq nc (add (nth 0 nc) 2))
                        (setq niveau (firstn (sub1 (length niveau))
                                             niveau))
                        (newr niveau nc)
                        (newr tree ")")
                        (newr tree (car arbre))
                        (quad-tree (cdr arbre) tree niveau))
                       (T(setq nc (last niveau))
                         (setq nc (add1 (nth 0 nc)))
                         (setq niveau (firstn (sub1 (length niveau))
                                              niveau))
                         (newr niveau nc)
                         (newr tree (car arbre))
                         (quad-tree (cdr arbre) tree niveau)))))))))
```

BIBLIOGRAPHIE

BIBLIOGRAPHIE

[ABI 85] **C. ABIGAIL**
Artificial intelligence and computer graphics.
Compt. Graphics World. Vol 8 (8) Augt. 85 pp. 11-20

[AIE 76] **L. AIELLO, M. AIELLO, G. ATTARDI, G. PRINI**
Recursive data types in LISP : a case study in type driven programming.
Actes du 2ème Col. Int. Sur la programmation
Paris Apr 76 pp. 232-248

[ALE 78] **N. ALEXANDRIS, A. KLINGER**
Picture decomposition, tree data-structures, and identifying directional symmetries as node communication
C. Graph and Im. Proc 8 (1978) pp. 43-77

[ALP 77] **ALPHALISP ® ALPHA MICRO SYSTEMS AM-100**
Alphalisp language programming system
User's Reference manual 1977

[ARY 84] **K.ARYA**
A functional approach to picture manipulation
Comp. Graphics Forum Vol. 3 (1) March 84 pp. 35-46

[AYA 85] **D. AYALA, P. BRUNET, R. JUAN, I. NAVAZO**
Object representation by means of nonminimal division quadtrees and octrees
ACM Trans-on Comp. Graphics Vol 4 (1) Jan 85 pp. 41-59.

[BAC 78] **J. BACKUS**
Can Programming be liberated from the VON-NEUMANN style , A functional style and its algebra of programs
CACM Vol 21 (8) Augt 78 pp. 613-641

[BAT 85] **D. BAILEY**
The University of Salford LISP/PROLOG System
S.P.E Vol 15 (6) June 85 pp. 595-609

[BAL 81] **D. H. BALLARD**
Strip trees : a hierarchical representation for curves
CACM vol 24 (5) May 81 pp. 310-321

[BAR 81] **W. BARTH**
The high-level graphics programming language PASCAL/GRAF
in : ENCARNACAO. J.L. Ed.
EUROGRAPHIGS 81 (1982) pp. 151-164

[BAS 84] **J.L BASILLE, S. CASTAN, J;Y; LATIL**
Structures parallèles en traitement d'images :
1er Col. Image Biarritz Mai 84 Vol 1 pp. 437-442

[BEN 75] **J.L. BENTLEY**
Multidimensional binary search trees used for associative searching
CACM Vol 18(9) Sept 75 pp.509-517

[BEZ 84] **J. BEZIVIN**
Simulation et langages orientés objets
BIGRE. Actes Journées AFCET sur les L.O.O
Nov. 84 pp. 1-18

[BIJ 81] **A. BIJAOUI**
Images et Informations : introduction au traitement numérique des images
Ed. MASSON Paris 81.

[BOA 81] **J.C. BOARDER**
Graphical programming for parallel processing systems
2nd. Int. Conf. on distributed Computing Systems
IEEE Vol 7 (81) pp. 467-475

[BOB 67] **D.G BOBROW, D.L. MURPHY**
Structure of a LISP system using two-level storage
CACM Vol 10 (3) March 67 pp. 155-159

[BOL 83] **H. BOLEY**
Artificial intelligence languages and machines
TSI Vol 2 (3) Mai-Juin 83 pp. 145-166

[BON 82] **P.R. BONO, J.L. ENCARNACAO, F.R.A HOPGOOD, P.J.W HENHAGEN**
GKS. The first graphics standard
IEEE Compt. Graphics and Appl. July 82 pp. 9-23

[BOY 75] **R.S. BOYER, J.S. MOORE**
Proving theorems about LISP functions
JACM Vol 22 (1) Jan 75 pp. 129-144

[BRA 75] **J.C. BRAID**
The synthesis of solids bounded by many faces
CACM Vol 18 (4) Apr 75 pp. 209-216

[CAS 84] **T. CASCIANI, B. FALCIDIENO, G. FASCIOLO, C. PIENOVI**
An algorithm for constructing a quadtree from polygonal regions.
C. Graphics Forum Vol 3 (4) Dec 84 pp. 269-274

[CAU 84] **R. CAUBET**
Elements de méthodologie dans la réalisation des logiciels graphiques
Aspects logiciels de base et logiciels d'application.
Thèse de Doct. d'Etat Toulouse Juin 84

[CHA 79] **S.K. CHANG, B.S. LIN, R. WALSER**
A generalized zooming technique for pictorial data base systems
AFIPS Conf. Proc. 1979 Nee Vol 48 pp. 147-156

[CHA 81] **S.K. CHANG, T.L. KUNII**
Dictorial data-base systems
The computer Vol 14 (11) Nov 81 pp. 13-21

[CHA 84-a] **J. CHAILLOUX**
La machine virtuelle LLM3
Rapport Projet VLSI INRIA Déc. 84

[CHA 84-b] **K. CHARROUF**
Etude de réalisation du système interactif PICASSO thèse
Thèse de Doctorat 3ème cycle USTL Lille 1 Déc. 84

[CHA 85] **J. CHAILLOUX**
LE-LISP de l'INRIA
Manuel de Référence Janv. 85

[CHE 84] **H.H. CHEN, S.Mc KUCK**
Combining relational and retwork retrieval methods
ACM.Proc.of Annual Meeting SIG MOD'84 June 84
pp 131-142

[CHO 81] **M. CHOCK, A.F. CARDENAS, A. KLINGER**
Manipulating data structures in pictorial information systems Computer Vol. 14(11) Nov. 81 pp. 43-50

[CHU 78] **W.L CHUNG**
Graphic language : semanties and code optimization
Proc. Int. Conf. Interactive Tech. in. Compt. Aided
Bologna Sept 78 pp. 255-264

[CLA 76-a] **J.H. CLARK**
Designing surfaces in 3D
CACM vol. 19(8) Augt 76 pp. 454-460

[CLA 76-b] **J.H. CLARK**
Hierarchical geometric models for visible surface algorithms
CACM Vol. 19(10) Oct. 76 pp. 547-554

[COH 79] **S. COHEN**
The A-table data-type for LISP systems
SIGPLAN Notices Vol. 14(10) Oct. 79 pp. 36-47

[COI 82] **P. COINTE**
Une réalisation de SMALL TALK en VLISP
TSI Vol. 1(4) Juillet-Aout 82 pp. 325-340

[COI 84-a] **P. COINTE**
Une implémentation des Coroutines en LISP, applications à SMALLTALK
Lectures Notes in Comput; Sciences
6th. Proc. Int. Symp; on Programmation Toulouse
Apr. 84 pp. 74-88

[COI 84-b] **P. COINTE**
Systèmes et langages orientés objets
Convention informatique Tome A' Paris Sept. 84
pp. 87-98

[COL 83] **A. COLMERAUER, H. KANOUI, M.V. CANEGHEM**
PROLOG, bases théoriques et développements actuels
TSI Vol. 2(4) Juillet-Aout 83 pp. 271-311

[COL 84] **J.F. COLONNA, NOWGLI**
De la visualisation de résultats de calcul à la création artistique
1ère col. Image BIARRITZ Mai 84 Vol. 1 pp.201-213

[COR 84] **V. CORDONNIER, M. MERIAUX**
Des architectures pour la synthèse d'images : une impérieuse nécessité
Convention Informatique Tome A Paris Sept. 84
pp. 43-47

[DAN 70] **P.E. DANIELSSON**
Incremental cuve generation
IEEE Trans. On. Compt. Vol. C-19(9) Sept. 70
pp. 783-793

[DAN 84] **P.E. DANIELSSON**
Algorithms and architecture for image processing
1ère Col. Image Biarritz Mai 84 pp. 461-472

[DEL 82] **C. DELOBEL, M. ADIBA**
Bases de données et systèmes relationnels
Ed. DUNOD Informatique 1982

[DUC 74] **A.DUCROT**
Principes d'organisation et de réalisation du système graphique interactif GIPSY
Thèse de 3ème cycle Université de Lille 1 1974

[DUF 81] **M.J.B. DUFF, S. LEVIALDI**
Languages and architectures for image processing
Academic Press Londres 1981

[DUR 83] **P. DURIF**
Etude d'una machine parallèle de synthèse d'images à découpage par objet
Thèse de 3ème cycle USTL Lille 1 Déc.83

[DYE 80] **C.R. DYER, A. ROSENFELD, H. SAMET**
Région représentation : Boundary Codes from quadtrees
CACM Vol. 23(3) March 80 pp. 171-179

[EBO 85] **M. EBOUEYA**
Sur l'intégration d'un SGBD relationnel aux outils de synthèse d'objets géométriques
Rapp. interne 35 85 LA 369 Université Lille 1 1985

[END 84-A] **G. ENDERLE, K. KANSY, G. PFAFF**
Computer graphics Programming
GKS. The graphic Standard
Springer Verlag 1984

[END 84-b] **G. ENDERLE**
GRS Implementation overview second edition
Compt. Graphics Forum Vol.3(2) June 84 pp. 181-189

[FER 83] **L. FERROUDJI**
Etude de tables de couleurs commutables pour un système de synthèse d'images
Mémoire d'ingénieur INI Oued Smar Alger 1983

[FER 85] **J. FERBER**
LISP : Langage de l'intelligence artificielle : Partie 2
MICRO-SYSTEMS Jan. 85 pp. 126-131

[FIN 74] **R.A. FINKEL, J.L. BENTLEY**
Quadtrees : a data structure for retrieval on composite keys
Acta informatica 4 1974 pp 1-9

[FOI 84] **L. FOISSEAU, J. CHOLVY**
Bases de données en CAO. Modélisations d'objets complexes et liaisons entre objets
MICAD'84 Paris Tome 3 pp. 944-957

[FOL 82] **J.D. FOLEY, A. VANDAM**
Fundamentals of interactive computer graphics
Addison-Wesley 1982

[FRI 78] **D.P. FRIEDMAN, D.S. WISE**
Aspects et applicative programming for file systems
IEEE Trans. on. Compt Vol. C-27(4) Apr.78 pp.289-296

[FUJ 83] **A. FUJIMOTO, K. INATA**
JAG Free images on a rasler CRT
Compt. Graphics Theory and Appl. Tokyo Apr. 83 pp. 2-15

[FU-82] **K.S.Fu, T.L. KUNIT**
Picture engineering
Springer Series in Information Sciences 1982

[GAN 81] **D. GANGA PADHYAY**
A franework for modelling graphic interactions
SPE Vol. 12(2) Feb. 81 pp. 141-151

[GHE 83] **M. GHERAB**
Visualisation en perspective d'objets mathématiques tridi mensionnels
Mémoire d'Ingénieur INI Oued Smer Alger 1983

[GOA 80] **G.B. GOATES, M.L. GRISS, G.J. HERRON**
PICTURE BALM : A LISP - Based graphics language system with flexible syntax and hierarchical data structure
C. Graphics Vol. 14(3) July 80 pp. 93-99

[GOL 83] **A. GOLDBERG, D. ROBSON**
SMALL-80 : The language and its implementation
Addison Wesley PÛblishing 1983

[GRA 80] **M. GRAVE**
Etude d'un noyau de système de synthèse d'image
Thèse de Docteur-Ingénieur USTL Lille I 1980

[GRE 77] **P. GREUSSAY**
Contribution à la définition interprétative et à l'implémentation des lambda-langages
Thèse d'Etat IPP Paris VI Nov. 1977

[GRI 77] **D. GRIES, N. GEHANI**
Some ideas on data types in high-level languages
CACM Vol. 20(6) June 77 pp. 414-420

[GRI 81] **M.L. GRISS, A.C. HEARN**
A portable LISP Compiler
SPE Vol. 11(6) June 81 pp. 541-605

[HAN 80] **P.R. HANAU, D.R. LENOROVITZ**
Prototyping and simulation tools for user/computer dialogue design
C. Graphics Vol. 14(2) Inly 80 pp. 271-278

[HAR 81] **M. HARDWICK, GMSG IBM UK Ltd**
Graphical data structures
C. Graphics Vol. 15(4) Déc. 81 pp. 376-404

[HEG 85-a] **G. HEGRON**
La technique du suivi de contour en synthèse d'image
TSI Vol. 4(4) Juillet-Aout 85 pp. 351-358

[HEG 85-b] **G. HEGRON**
Synthèse d'images : algorithmes élémentaires
Ed. DUNOD Informatique 1985

[HEW 84] **W.T. HEWITT**
PHIGS. Programmer's hierarchical interactive graphic system
C. Graphics Forum Vol. 3(4) Déc. 84 pp. 299-300

[HON 83] **U. HONSCHOPP, W.M. LIPPE, F. SIMON**
Compiling functional languages for VON-NEUMANN machines
SIGPLAN Notices Vol. 18(6) June 83 pp.22-27

[INR 83] Bulletin de liaison de la recherche en informatique et automatique
INRIA N° 88 1983

[JAC 80] **C.L. JACKINS, S.L. TANIMOTO**
Octrees and their use in representing three-dimensional objets
C. Graphics and Image Proc. 14-1980 pp. 249-270

[JOR 73] **C.R. JORDAN**
A note on LISP universal S. Functions
Computer Journal Vol. 16(2) May 73 pp. 124-125

[JOU 84-a] **M. JOURDAN**
Les grammaires attribuées : implantations, applications, optimisations
Thèse de Docteur-Ingénieur Paris VII Mai 1984

[JOU 84-b] **P. JOUVELOT, D. LECONTE DES FLORIS**
Intelligence artificielle et systèmes experts : le langage LISP
Minis et Micros n° 208 1984 pp. 45-48

[KAM 83 **A. KAMRAN, M.B. FELDMAN**
Graphics programming independant of interaction techniques and style
†. Graphics Vol. 17(1) Janv. 83 pp. 58-66

[KIJ 83-a] **S. KIJNER**
Vision assistée par ordinateur "VAO" Partie 1
Micros-Systems n° 36 Nov. 83 pp. 114-135

[KIJ 83-b] **S. KIJNER**
Vision assistée par ordinateur "VAO" Partie 2
Micro-Systems n° 37 Déc. 83 pp. 126-142

[KLI 76] **A. KLINGER, C.R. DYER**
Experiments in picture representation using regular decomposition
Compt. Graphics and Ima. Pro. 5 1976 pp.68-105

[KLI 78] **A. KLINGER, M.L. THODE, V.T.TO**
Accessing image data
Int'l. J. Poliay Analysis and information system
Vol. 1(2) Jan. 78 pp. 171-189

[KUL 68] **H.E. KULSRUD**
A general purpose graphic language
CACM Vol. 11(4) Apr. 68 pp. 247-254

[KUN 74] **T.L. KUNII, J.M. TENEBAUM**
A relational data-base schema for describing complex pictures with color and texture
Proc. Second Int'l Joint Conf. on Pattern. Recogn. Aug 74 pp. 310-316

[LAC 84] **G. LACOSTE, B. GUIMBAL, T. BRIZARO**
Language de haut niveau pour images Sonar
Premier Coll. Image Biarritz Mai 1984 Vol. 2 pp-931-936

[LAN 65] **P.J. LANDIN**
A correspondance between ALGOL 60 and Church's lambda notation
CACM Vol. 8(2) Augt. 65 pp.89-101

[LEA 74] **B.M. LEAVENWORTH, J.E. SAMMET**
An overview of non-procedural languages
SIGPLAN Vol. 9(4) Apr. 74 pp.1-12

[LED 77] **H.F. LEDGARD, R.W. TAYLOR**
Two views of data abstraction
CACM Vol. 20(6) June 77 pp. 382-384

[LEV 79] **J.J. LEVY**
Propriétés syntaxiques du λ-calcul
Rapport interne LITP Université Paris VII Fev. 79

[LIG 80] **P. LIGNELET, P. COINTE**
LISP Manuel d'initiation
Notes de cours IPP Paris VI 1980

[LON 85] **R.L. LONDON, R.A. DUISBERG**
Animating Programs using SMALLTALK
The computer vol. 18(8) Augt 85 pp. 61-71

[LOR 82] **R. LORIE**
Issues in data bases for design applications file structures and data bases for CAD
Ed. J. LENCARNACAO and F.L. KRAUSE
N.H. Amsterdam 1982 pp. 213-222

[LUC 77] **M. LUCAS**
Contribution à l'étude des techniques de communication graphique avec un ordinateur; Elements de base des logiciels graphiques interactifs. Thèse de Docteur-Es. Sciences INPG Grenoble Déc. 77

[LUC 84] **M. LUCAS**
La synthèse d'images par ordinateur
Premier colloque Image Biarritz Mai 1984 Vol. 1 pp.11-19

[MAC 84-a] **C. MACHOVER, W. MYERS**
Interactive Computer Graphics
The Computer Vol. 17(10) Oct. 84 pp. 145-161

[MAC 84-b] **MACLISP (MULTICS)**
Manuel de Référence 1984

[MAI 83] **H. MAITRE, A. GROUZET, P. LERAY, D. NASSE, J.P. TEMIME, J. DELVILTE**
Les tendances actuelles en traitement d'images
L'écho de la Recherche n° 113 3ème trimestre 83
pp. 31-44

[MAI 84] **H. MAITRE**
Banque d'images et bases de données images
1er Coll. Image Biarritz Mai 84 Vol. 2 pp. 905-916

[MAN 82] **M. MANTYLA, R. SULONEN**
GWB : a solid modeler with enter operations
IEEE Compt. Graphics and App. Vol 2(7) Sept. 82 pp. 17-31

[MAR 79] **J. MARTI, A.C. HEARN, M.L. GRISS, C. GRISS**
Standard LISP report
SIGPLAN Vol 14(10) Oct. 79 pp. 48-68

[MAR 82] **F. MARTINEZ**
Vers une approche systématique de la synthèse d'image
Aspects logiciels et matériels
Thèse d'état INPG Grenoble 1982

[MCC 78-a] **J. Mc CARTHY**
A micro-manual for LISP-NOT The whole truth
SIGPLAN Vol. 13(8) Augt 78 pp. 215-216

[MCC 78-b] **J. Mc CARTHY**
History of LISP
SIGPLAN Vol. 13(8) Augt 78 pp. 217-223

[MEA 82] **D. MEAGHER**
Geometric modelling using octree encoding
C. Graphics and Image Proc. 19(1982) pp.129-147

[MER 79] **M. MERIAUX**
Etude et réalisation d'un terminal graphique couleur tridimensionnel fonctionnant par taches
Thèse de Doct. Ingénieur Université de Lille I Jan. 79

[MER 84] **M. MERIAUX**
Contributions à l'imagerie informatique : aspects algorithmiques et architecturaux
Thèse d'Etat Université de Lille I 1984

[MIN 84] Informatique graphique et images
MINI-MICROS Numéro Hors Série 204 bis 1984

[MIR 84] **S. MIRANDA, J.M. BUSTA**
L'art des bases de données
Tome 1 : Introduction aux bases de données
Ed. Eyrolles 1984

[MOR 76] **P. MORVAN, M. LUCAS**
Images et ordinateur : Introduction à l'infographie interactive
Ed. Larousse Série Informatique 9-76

[MUL 83] **MULTICS.** Common Commands
Brochure Technique Apr. 83

[NEW 79] **W. M. NEWMAN, R.F. SPROULL**
Principles of interactive Computer graphics
2nd. Ed. Mc Graw Hill Int. Student Ed NY. 1979

[OBR 75] **C.D. O'BRIEN, H.G. BOWN**
Image : an language for the interactive manipulation of graphics environment
Proc. of. SIGGRAPH 75 Compt. Graphics Vol. 9(1) 1975 pp.70-77

[OCA 72] **J.F. O'CALLAGHAN**
Use of a picture language to generate descriptions of line drawings in an interactive system
in Graphics language IFIP 1972 Ed. F. NAKE, A. ROSENFELD
pp. 123-143

[OHL 80] **M.R. OHLSON**
Design language for interactive graphics applications Systems
University Micro Films Int. Augt. 80 pp. 1-47

[OKE 83] **R.A. O'KEEFE**
PROLOG Compared with LISP ?
SIGPLAN Vol. 18(5) May 83 pp. 46-56

[OLI 83] **M.A. OLIVIER, N.E. WISEMAN**
Operations on quadtree encoded images
The Computer Journal Vol. 26(1) 1983 pp. 83-91

[OLI 84-a] **M.A. OLIVIER**
Two display algorithms for octrees
Eurographics 84 Copenhagen Sept 84 pp. 251-264

[OLI 84-b] **M.A. OLIVIER, T.R. KING, N.E. WISEMAN**
Quadtree scan conversion
Eurographics 84 Copenhagen Sept. 84 pp 265-276

[OMO 79] **J. OMOLAYOLE, A. KLINGER**
A hierarchical data structure scheme for storing pictures
Tech. Report. Compt. Sciences Dept. UCLA 1979

[PAR 82] **C. PARENT**
Etude et spécifications d'un éditeur d'images : application à la bureautique
Thèse de Doct. 3ème cycle USTL Lille I 1982

[PEL 84] **M. PELERIN**
Algorithmes graphiques parallèles
1er coll. Image Biarritz Mai 84 Vol. 2 pp. 613-619

[PEL 85] **M. PELERIN**
Synthèse d'images et parallélisme : algorithmes et architectures
Thèse de Doct. Ingénieur USTL Lille I Janv. 85

[PRE 82] **K. PRESTON Jr. L. UHR**
Multicomputers and image processing : algorithms and programs.
Notes and reports in Compt. Sciences and Applied mathematics
Ed. Academic Press 1982

[QUE 83] **C. QUEINNEC**
Langage d'un autre type : LISP
Ed. Eyrolles Paris 1983

[RAE 85] **G. RAEDER**
A survey of current graphical programming techniques
The computer Vol. 18(8) Augt 85 pp. 11-25

[RED 84] **M. REDJIMI**
Etude et réalisation d'un système parallèle pour le traitement graphique
Thèse de Doct. Ingénieur USTL Lille I Sept. 84

[REY 70] **J.C. REYNOLDS**
GEDANKEN - A simple typeless language based on the principle of completeness and the reference concept
CACM Vol. 13(5) May 70 pp. 308-319

[RIE 85] **D. RIEU-RIALHE**
Modèle et fonctionnalités d'un SGBD pour les applications CAO
INRIA Journées Bases de données avancées MARS 1985

[ROB 66] **L.G. ROBERTS**
A graphical service system with variable syntax
CACM Vol. 9(3) March 66 pp. 173-175

[ROB 70] **M.F. ROBBINS, J.D. BEYER**
An interactive Computer System using ggraphical flowchart input
CACM Vol. 13(2) Feb 70 pp. 115-119

[ROT 80] **H. ROTHLISBERGER**
Modèles de systèmes graphiques interactifs et applications à la conception d'une station de mise en pages de textes composites
Thèse de Docteur Es.Sciences EFPL LAUSANNE 1980

[ROW 83] **L.A. ROWE, K.A. SHOENS**
Programming language constructs for screen definition
IEEE Trans. on Soft. Eng. Vol SE-9(1) Jun. 83 pp.31-39

[SAM 80] **H. SAMET**
Région représentation : quadtrees from boundary codes
CACM Vol. 23(3) March 80 pp. 163-170

[SHA 70] **A. SHAW**
Parsing of graph representable picture
JACM Vol. 17(3) July 70 pp. 453-481

[SHA 75] **L.G. SHAPIRO**
ESP : a high level graphics language
Proc. of. SIGGRAPH 75. C. Graphics Vol. 9(1) 1975 pp. 70-75

[SHA 79] **A. SHAW**
Basic algorithms and programming languages
Ecole d'été informatique CEA. IRIA. EDF France Jul. 79

[SHO 79] **J.F. SHOCH**
An overview of the programming language SMALLTALK-72
SIGPLAN Vol. 14(9) Sept. 79 pp. 64-73

[SHR] **G. SHRAK**
Design, implementation and experiences with a higher-level graphics language for interactive computer-aided design purpose
C. Graphics Vol. 10(1) 1976 pp. 10-17

[SIG] "COMPUTER GRAPHICS" A quartely report of SIGGRAPH-ACM
Vol. 13(3) 1979

[SLA 84] **M. SLATER**
GKS in PASCAL
C. Graphics FORUM Vol. 3(4) Déc. 84 pp. 259-268

[SLI 77] **Y. SLIMANI, Y. CHAHER**
Vers une définition formelle des langages de programmation
Rapport de recherche CRIN Nnacy Janv. 77

[SLI 84] **Y. SLIMANI**
Leslangages en intelligence artificielle
Pub. interne Université Es. Senia Algérie Mai 1984

[SLI 85] **Y. SLIMANI**
Comparaison entre les langages VON-NEUMANN et les langages non-procéduraux - Exemple des langages applicatifs
Pub. interne Université Es. Senia Algérie Mai 1985

[SPO 84] **D.L. SPOONER**
Database support for interactive computer graphics
ACM Proc. of Annual Meeting SIGMOD'84 June 84 pp. 90-99

ΣPR 82] **R.F. SPROULL**
Using program transformations to derive line-drawing algorithms
ACM. Trans on graphics Vol. 1(4) Oct. 82 pp. 259-273

[STI 76] **M. STIES, B. SANYAL, K. LEIST**
Organization of object data for an image information systel
Proc. third Int'l Joint Conf. Pattern Recognition 1976 pp. 863-869

[SUS 81] **G.J. SUSSMAN, J. HOLLOWAY, g.l. steel jr, A. BELL**
Scheme-79 - LISP on a chip
The Computer Vol. 14(7) July 81 pp. 10-21

[SUZ 83] **Y. SUZUKI**
Implementation of the graphical kernel system (GKS)
C. Graphics Theory and aPP. tOKYO apr. 83 pp. 67-74

[TAK 72] **Y. TAKASANA, S. MORIGUCHI, T. SAKAMAKI**
A graphical manipulating languages
Graphics Languages IFIP N. Holland 1972
Ed. F. NAKE, A. ROSENFELD pp. 327-333

[TAK 83] **I. TAKEUCHI, H. OKUNO, N. OHSATO**
TAO : a harmonic mean of LISP, PROLOG, and SMALLTALK
SIGPLAN Vol. 18(7) July 83 pp. 65-74

[TES 73] **L. TESLER, H.J. ENEA, D.C. SMITH**
The LISP 70
Pattern Matching System Proc. IJCAI 3 1973

[THA 81] **N.M. THALMANN, D. THALMANN**
A graphical Pascal extension based or graphical types
SPE Vol. 11(1) Janv. 81 pp. 53-62

[WAR 77] **D.H.D. WARREN, L.M. PEREIRA, F. PEREIRA**
PROLOG : the language and its implementation compared with LISP
SIGPLAN/SIGART Vol. 12(8) Augt. 77 pp. 109-115

[WEG 76] **P. WEGNER**
Programming languages - the first 25 years
IEEE Trans. on Compt. Vol C-25(12) Déc. 76 pp. 1207-1225

[WEL 80] **D. WELLER, R. WILLIAMS**
Graphic and relational data base support for problem solving tubrial and selecting readings in interactive compt graphics
Feb. 80 pp. 193-199

[WEL 84] **D.L. WELLER, B.W. YORK**
a RELATIONAL REPRESENTATION OF AN ABSTRACT TYPE SYSTEM
IEEE Trans. on Soft. Eng Vol SE.10(3) May 84 pp. 303-309

[WIL 71] **R. WILLIAMS**
A survey of data structures for computer graphics systems
ACM Compt. Surveys Vol 3(1) March 71 pp. 1-21

[WIL 74] **R. WILLIAMS**
On the application of relational data structures in computer graphics
Proc. IFIP 74 Vol. 4 Augt. 74 pp. 723-726

[WOO 84] **T.C. WOO**
Interfacing solid modeling to CAO and CAM : data structures and algorithms for decomposing a solid
Computer Vol. 17(12) Déc. 84 pp. 44-49

[YAU 83] **M.M. YAU, S.N SRIHARI**
A hierarchical data structure for multidimensional digital images
CACM Vol. 26(7) July 83 pp.504-515

[YIP 84] **C.K. YIP**
The pascal graphics system
SPE Vol. 14(2) Feb. 84 pp. 101-118

[ZAV 82] **P. ZAVE**
An operational approach to requirements specification for embedded systems
IEEE Trans. on Soft. Eng Vol. SE-8(3) May 82 pp. 250-269

RESUME

La synthèse d'images fait partie des applications informatiques pour lesquelles on essaie de mettre en oeuvre des architectures et des logiciels répondant, le mieux possible, à deux contraintes principales qui sont l'espace mémoire et le temps de calcul.

Devant cette situation, et après avoir analysé les besoins du graphique en matière de structures de données et d'algorithmique, notre travail est une approche par l'utilisation des langages applicatifs (type LISP) en graphique, sous deux aspects :

i) structure de donnée où après avoir mis en évidence les relations existantes entre structures de données et systèmes graphiques, nous proposons un nouveau modèle de structure de donnée susceptible d'être utilisé pour représenter les objets graphiques, à savoir la structure de liste

ii) algorithmique, où après une présentation de l'approche de programmation non-procédurale, nous montrons comment un des concepts fondamentaux des langages applicatifs (la récursivité) peut contribuer à une expression naturelle d'algorithmes graphiques.

MOTS CLES

Systèmes graphiques, structures de données, algorithmique graphique, langagees graphiques, langages non-procéduraux, langages applicatifs, LISP.