Dans le cadre de CLOVIS : un logiciel pilote du terminal de synthèse d'images réalistes HELIOS

Alain Boulze

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

CENTRE AGREE
DE GRENOBLE (C.U.E.F.A)

MEMOIRE

présenté en vue d'obtenir

le diplôme d'Ingénieur

en

INFORMATIQUE

par

Alain BOULZE

Dans le cadre de CLOVIS : un logiciel pilote
du terminal de synthèse d'images réalistes HELIOS

SOUTENU LE : 29 Octobre 1984

JURY

Président : Monsieur le Professeur RANCHIN J.Y

Membres : Monsieur le Professeur BOLLIET L.
Mrs MARTINEZ F.
MIRIBEL J.F
Mme SARRASIN M.T

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

CENTRE AGREE
DE GRENOBLE (C.U.E.F.A)

MEMOIRE

présenté en vue d'obtenir

le diplôme d'Ingénieur

en

INFORMATIQUE

par

Alain BOULZE

Dans le cadre de CLOVIS : un logiciel pilote
du terminal de synthèse d'images réalistes HELIOS

SOUTENU LE : 29 Octobre 1984

JURY

Président : Monsieur le Professeur RANCHIN J.Y

Membres : Monsieur le Professeur BOLLIET L.
Mrs MARTINEZ F.
MIRIBEL J.F
Mme SARRASIN M.T

Le travail présenté dans ce mémoire a été réalisé dans le cadre du projet CLOVIS, au sein de l'équipe "Images" du laboratoire ARTEMIS.

Je tiens à remercier chaleureusement :

Mr RANCHIN du Conservatoire National des Arts et Métiers qui m'a fait l'honneur d'accepter mon dossier de mémoire,

Mr BOLLIET pour l'aide qu'il m'a apportée tout au long du cycle C, et qui a bien voulu présider ce jury,

Mr MARTINEZ pour l' accueil dans son équipe cette dernière année et les conseils qu'il m'a prodigués dans la direction de mes travaux,

Mme SARRASIN, ingénieur dans une unité de HEWLETT-PACKARD et ayant fait partie de l'équipe "Communication Graphique" de l'IMAG voici deux ans, et Mr MIRIBEL responsable du département "Robotique et Graphique" de la société ITMI, qui ont accepté de siéger à ce jury.

Que soient remerciés aussi les membres de l'équipe "Images" pour leur disponibilité et leurs encouragements, et en tout premier lieu J.F. GRABOWIECKI, avec qui j'ai collaboré tout spécialement, ainsi que P.GENOUD, V.ZARATE, K.CHIBANE, de même que P.BOULLE, participant à la vie de l'équipe, bien que d'une société extérieure.

Je n'oublierai pas l'ensemble de l'équipe graphique du laboratoire ARTEMIS, ainsi que Madame CHALAND, secrétaire de Mr BOLLIET, pour son aide devant les problèmes administratifs.

Enfin, je remercie les personnes qui se sont occupées du tirage de ce document.

TABLE DES MATIERES

INTRODUCTION..... 1

CHAPITRE I : GENERALITES SUR LA SYNTHESE D'IMAGES

1. Notions de base..... 3
  1.1. Les différents types d'images..... 3
  1.2. Classification des informations de base..... 4

2. Les grandes étapes de la synthèse d'images..... 5
  2.1. Description de la scène..... 6
  2.2. Contruction de la scène..... 6
    2.2.1. Modélisation de la visibilité..... 6
    2.2.2. Modélisation de l'aspect des objets..... 7
  2.3. Synthèse finale de l'image..... 8
    2.3.1. Remplissage de taches..... 8
    2.3.2. Application des textures..... 8

3. Les différentes technologies..... 8
  3.1. Dispositifs d'affichage..... 9
  3.2. Dispositifs d'entretien..... 9
  3.3. Dispositifs de coloration..... 9
  3.4. Exemples de systèmes..... 10
    3.4.1. Systèmes minimaux..... 10
    3.4.2. Systèmes généraux..... 10

CHAPITRE II : ORGANISATION D'UN SYSTEME DE SYNTHESE D'IMAGES -

ASPECTS CONCEPTEUR ET TECHNIQUE

1. Elements de base..... 13
  1.1. Les opérations..... 13
    1.1.1. Description de la maquette..... 13
    1.1.2. Construction de la maquette..... 13
    1.1.3. Prise de vue..... 14
    1.1.4. Affichage..... 14
    1.1.5. Récapitulatif..... 14
  1.2. Les processus..... 14
    1.2.1. L'attribution..... 15
    1.2.2. La consultation..... 15
    1.2.3. La visualisation..... 15
    1.2.4. La description..... 16

2. Organisation générale du système..... 17
  2.1. La notion de synthétiseur..... 17
  2.2. Influence de la configuration matérielle..... 17
    2.2.1. Console bas de gamme..... 17
    2.2.2. Console évoluée..... 18

2.2.3. Connexion à un calculateur satellite..... 19
2.2.4. Configuration multiple..... 19
2.3. Organisation interne d'un synthétiseur..... 20
2.3.1. Cas particulier: synthètiseur monoprocessus..... 20
2.3.2. Organisation d'un synthétiseur multiprocessus..... 21

3. Les performances d'un système..... 24

CHAPITRE III : EXEMPLE DE SYSTEME - ASPECTS MATERIEL ET LOGICIEL

1. Présentation générale du système..... 25

2. Les synthétiseurs cablés : le prototype HELIOS..... 27
2.1. La configuration matérielle d'HELIOS..... 28
2.2. L'architecture générale..... 29
2.2.1. Organisation hiérarchique..... 29
2.2.2. Les structures de données..... 29
2.3. Originalité d'HELIOS : la notion de face plane..... 30
2.4. Les différents processeurs..... 31
2.4.1. Processeur de visibilité..... 31
2.4.2. Processeur des textures..... 32
2.4.3. Processeur de réflexion..... 33
2.4.4. Processeur d'éclairage et d'affichage..... 35
2.5. Les différentes versions d'HELIOS..... 36
2.5.1. Utilisation de microprocesseurs..... 36
2.5.2. Arcnitecture banalisée..... 37
2.5.3. Autres approches..... 38

3. Les synthétiseurs programmables : CLOVIS..... 38
3.1. Le logiciel pilote..... 38
3.2. Les principaux logiciels de base graphiques existants..... 40
3.2.1. Structure des logiciels de base..... 40
3.2.2. Présentation de quelques logiciels..... 41
3.3. Le logiciel principal de CLOVIS..... 42
3.3.1. L'unité de communication..... 42
3.3.2. L'unité de contrôle..... 49
3.4. Conclusion..... 50

CHAPITRE IV : PRESENTATION DU LOGICIEL A REALISER -

ASPECTS UTILISATEUR ET SYSTEME

1. Cadre de réalisation du logiciel..... 51
1.1. Configuration matèrielle..... 51
1.2. Objectifs du logiciel..... 52

2. Point de vue de l'utilisateur..... 53
2.1. Liste et classification des types d'attributs..... 53
2.1.1. Morphologie..... 53
2.1.2. Aspect..... 55
2.1.3. Géométrie..... 56
2.1.4. Eclairage..... 56

2.1.5. Géométrie de la prise de vue..... 56
2.1.6. Géométrie de l'affichage..... 57
2.1.7. Récapitulatif..... 57
2.2. Liste des primitives et commandes..... 58
2.2.1. Les primitives de structuration..... 58
2.2.2. Les primitives d'attribution..... 59
2.2.3. Les primitives de consultation..... 62
2.2.4. Les primitives de visualisation..... 63
2.2.5. Les primitives de description..... 63
2.2.6. Primitives annexes..... 63
2.2.7. Tableau récapitulatif des commandes..... 64

3. Point de vue du concepteur..... 65
3.1. Gestion de la structure de données..... 65
3.1.1. Présentation d'un noeud de l'arborescence..... 65
3.1.2. Principes généraux de fonctionnement..... 67
3.1.3. Automate d'analyse..... 67
3.1.4. Primitives et structures de données internes associées.. 69
3.1.5. Gestion du nombre de noeuds touchés..... 73
3.1.6. Calcul du barycentre..... 74
3.1.7. Conclusion..... 74
3.2. Gestion des processus..... 74
3.2.1. Facteurs influant la gestion des processus..... 75
3.2.2. Mémorisation des informations..... 75
3.2.3. Solutions retenues..... 76

4. Conclusion..... 78

CHAPITRE V : REALISATION LOGICIELLE - ASPECT TECHNIQUE

1. Présentation des matériels..... 81
1.1. Le calculateur principal..... 81
1.2. Le calculateur satellite..... 82

2. Présentation du logiciel..... 83
2.1. Description des données..... 83
2.1.1. Les descripteurs d'attributs..... 83
2.1.2. Les types d'attributs et leur codage..... 91
2.2. Description des modules..... 92
2.2.1. Interface système/application..... 93
2.2.2. Modules système..... 93
2.2.3. Interface système/matériel..... 94
2.2.4. Structuration générale..... 94

3. Réalisations dans le cadre de l'unité de contrôle..... 96
3.1. Gestion des processus..... 96
3.1.1. Fonctions des modules COMUNIV et DESATTS..... 96
3.1.2. Initialisations des structures de données associées..... 101
3.2. Liaison avec HELIOS..... 103
3.2.1. Logiciel pilote d'HELIOS..... 104
3.2.2. Echanges VAX-HELIOS..... 105

4. Réalisations dans le cadre de l'unité de communication..... 110
4.1. Gestion du descripteur matériel..... 110
4.1.1. Quelques précisions sur les faces d'HELIOS..... 110
4.1.2. Rappel du descripteur..... 111
4.1.3. Règles de gestion..... 111
4.2. Utilisation du lien objet/faces..... 115
4.2.1. Influence sur les processus de visualisation et description..... 115
4.2.2. Influence sur les processus d'attribution et consultation..... 115

5. Réalisations dans le cadre de l'unité de description-visualisation..... 116
5.1. Les transformations géométriques..... 116
5.1.1. Utilisation de coordonnées homogènes..... 116
5.1.2. Règles de composition..... 116
5.2. Fenêtre et cloture..... 118
5.2.1. Définitions de base..... 118
5.2.2. Descripteur d'attribut..... 120
5.2.3. Le clipping en 2D..... 121
5.3. Description d'une couleur..... 122
5.3.1. Affichage d'une palette..... 123
5.3.2. Identification d'une couleur de base..... 124
5.3.3. Description de la couleur finale..... 124
5.4. Description de l'éclairage..... 127
5.4.1. Description de la couleur de la source lumineuse..... 127
5.4.2. Description de la direction de la source..... 127
5.4.3. Description de la lumière ambiante..... 129

6. Exemple d'utilisation du logiciel..... 129

CONCLUSION..... 131

ANNEXES

Annexe 1: vers une architecture banalisée des synthétiseurs d'images..... 135
Annexe 2: quelques exemples de structures graphiques arborescentes. 143
Annexe 3: utilisation des coordonnées homogènes..... 169
Annexe 4: modèlisation de l'aspect couleur des objets..... 175
Annexe 5: exemple d'utilisation..... 181

BIBLIOGRAPHIE..... 195

INTRODUCTION

Point n'est besoin de citer ici les banalités d'usage pour affirmer que l'image a été, est et sera un considérable outil de communication pour l'homme. Gravures préhistoriques, peintures, photographies, films et télévision sont là pour attester de cette vérité, et ce, tout au long des âges.

Plus récemment, l'avènement pour informaticiens et utilisateurs de l'outil informatique, d'abord des consoles de visualisation , puis du graphique pour représenter des phénomènes physiques, suivre des processus industriels , ou encore faciliter la compréhension de résultats atteste de l'intérêt croissant des techniques graphiques. Cette évolution se poursuit depuis quelques années par l'essor de l'image, et de sa création par ordinateur, dans une société marquée par l'explosion de l'audio-visuel.

La synthèse d'images a eu d'ailleurs, en cette année 1984, une sorte de double consécration, par l'organisation d'une part du 1er colloque image à Biarritz essentiellement réservé aux initiés, par la publication d'autre part d'un numéro spécial de la revue 'Sciences et Techniques' (SET84) en mai de cette année, consacré donc à ce sujet et ouvert à un plus large public.

Ce colloque a été du reste l'occasion de constater l'extraordinaire gamme d' applications de ce domaine, y compris en France, alors que les USA y faisaient figure d' "ogres". Synthèse d'images et simulation de robot -sujet développé par le laboratoire LIFIA de l'IMAG-, simulateurs d'entrainement, imagerie médicale, perspectives du videodisque, création artistique, production industrielle de dessin animé par ordinateur, sans oublier la CAO ont été quelques-uns des thèmes exposés au fil des différentes conférences, et prouvent, si besoin en était, toute la richesse de ce que peut apporter la synthèse d'images.

Ce mémoire a donc comme cadre cette discipline, passionnante et pleine d'avenir , faisant suite à un travail de synthèse sur ce sujet exposé lors de l'examen probatoire du cycle Ingénieur du C.N.A.M. (BOU83).

Il présente la conception et la réalisation d'un logiciel pilote du terminal interactif de synthèse d'images réalistes en temps réel HELIOS, développé dans l'équipe 'Image' du laboratoire ARTEMIS de l'IMAG.

Les idées maitresses de ce travail peuvent se résumer en quelques mots-clés:

- manipulation d'OBJETS
- INTERACTIVITE maximale
- EXPLOITATION des possibilités du MATERIEL

- COMPLEMENTARITE par LOGICIEL des actions du terminal

Les soucis qui ont guidé cette étude sont lisibilité, portabilité et fiabilité du logiciel en raison de sa préoccupation "universelle" -adaptation au maximum d'applications et de matériels- ainsi que d'une possible commercialisation.

Ce mémoire s'articule autour de cinq parties.

Le chapitre I expose de facon succinte les principales notions sur la synthèse d'images.

Le chapitre II présente sous un aspect quelque peu théorique l'organisation d'un système de synthèse d'images. Suit l'exposé d'un exemple d'un tel système, à savoir le projet logiciel CLOVIS et le terminal HELIOS, faisant l'objet du chapitre III.

Les chapitres IV et V entrent en détail dans le logiciel réalisé, dans ses aspects utilisateur et système d'abord, sur un plan plus technique ensuite.

Enfin, nous concluons par l'analyse des résultats et les voies d'avenir qui s'ouvrent à la suite de cette réalisation.

CHAPITRE I : GENERALITES SUR LA SYNTHESE D'IMAGES

La synthèse d'images est une des deux techniques de l'Infographie Interactive -"Computer Graphics"- (LUC77), la seconde étant l'analyse d'images ou reconnaissance des formes.

Ce chapitre nous permettra de rappeler succintement les points généraux essentiels de cette technique, à savoir :

- les concepts de base : terminologie, différents types d'images, classification des informations.
- les grandes étapes de la synthèse d'images.
- les diverses technologies utilisées.

L'étude réalisée dans le cadre de l'examen probatoire du CNAM (BOL83) permet une approche plus précise de tous ces problèmes et une bibliographie plus dense pour toutes références.

1. Notions de base

Les concepts de base développés par F.MARTINEZ (MAR82), et M.LUCAS (LUC77), ont permis de dégager des éléments de structuration des logiciels graphiques interactifs, par des essais de regroupement et classification des images et des informations véhiculées par elles.

1.1. Les différents types d'images

Deux critères de classification sont envisagés :

- technique picturale
- nature de l'information véhiculée

Le premier mode de classification permet de distinguer les dessins au trait, constitués d'un ensemble de points ou segments, définis par un couple de coordonnées, et auxquels est associé un graphisme définissant couleur, taille, texture, épaisseur ..., et les dessins point par point, dont l'élément final est le pixel exprimé dans un espace (Rouge, Vert, Bleu), ce type de dessins recoupant alors la notion d'images.

Par le second mode se dégagent quatre types d'images :

- abstraites: pas d'information tangible

- symboliques: informations quantitatives

- figuratives: représentation simplifiée d'objets

- réalistes: représentation fidèle d'objets

1.2. Classification des informations de base

Dans le cadre des images -ensembles de pixels- réalistes -rendu d'une fidèlité maximale-, celles-là mêmes qui nous intèressent dans cette étude, les informations peuvent être éclatées en six classes, elles mêmes réparties en trois catégories. La figure I.1 schématise cette classification.

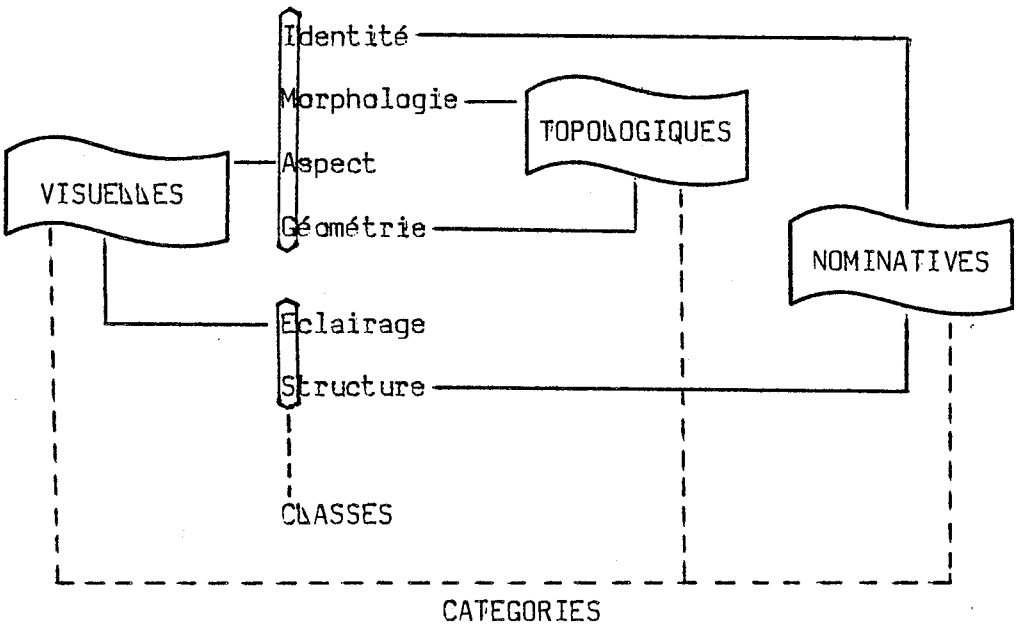


Figure I.1: Classification des informations véhiculées par une image réaliste

Il est à remarquer, dès cette présentation générale que ces concepts de base se retrouveront tout au long de cette étude, et imprimeront une marque très importante sur la réalisation du logiciel.

Explicitons rapidement ces notions:

- Classe IDENTITE: nomination des objets ou ensembles d'objets, ou scènes

- Classe STRUCTURE: relation des objets entre eux telle appartenance, composition
  * ces deux classes formant le groupe des informations NOMINATIVES, non présentes sur l'image, mais indispensables à l'utilisateur

- Classe MORPHOLOGIE: forme intrinsèque de chaque objet

- Classe GEOMETRIE: positionnement des objets les uns par rapport aux autres
  * ces deux classes contenant des informations TOPOLOGIQUES, définissant domaine et visibilité des objets

- Classe ASPECT: aspect intrinsèque de chaque objet

- Classe ECLAIRAGE: description de la, ou des sources lumineuses
  * ces deux classes constituant le groupe des informations VISUELLES, dont l'incidence se fait ressentir en chaque point de l'image.

Autre subdivision importante que l'on retrouvera également présente lors de la présentation du logiciel, la notion de TYPE d'informations à l'intérieur de chaque classe, définissant de facon précise la nature de l'information -par exemple les types VECTEUR, CERCLE, ... dans la classe Morphologie-.

Enfin, notons que les classes I, M, A, G se rapportent à un objet, alors que les classes E et S interviennent sur un ensemble d'objets, autrement dit une scène.

2. Les grandes étapes de la synthèse d'images

L'image synthétisée est la transcription par ordinateur selon les directives du seul concepteur d'une scène, idée d'ensemble voulue par un utilisateur, image analysée en dernier lieu par l'opérateur humain d'où résulte la scène cette fois percue d'une certaine facon.

La figure I.2 montre les interactions entre l'opérateur humain et l'ordinateur, ou d'une facon plus générale, le système de synthèse, au cours du processus complet de synthèse.

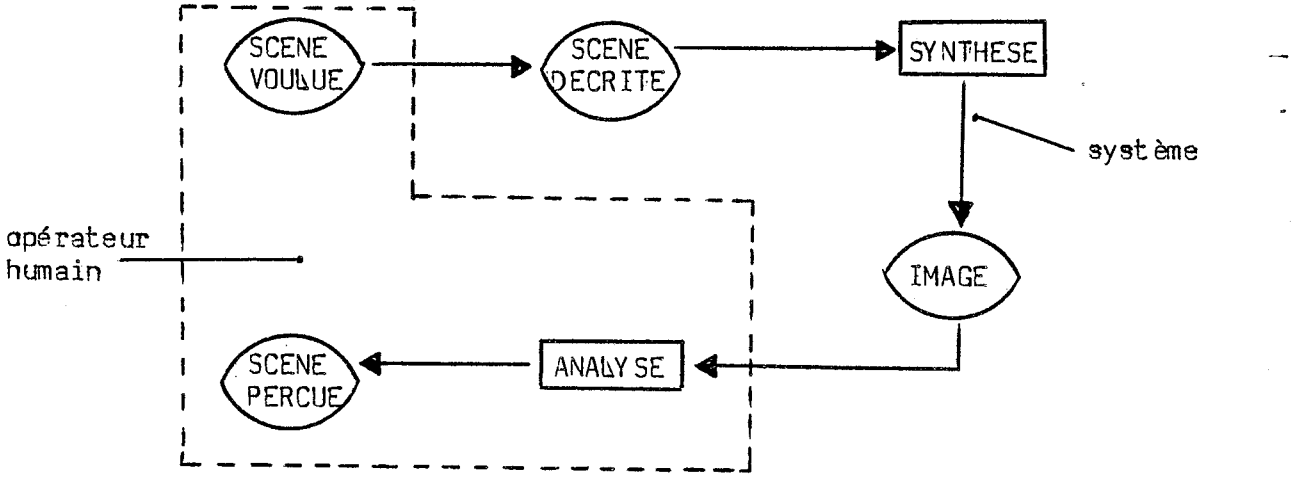


Figure I.2: Schématisation de la synthèse d'images

Par ce schéma, sont mises en évidence les trois grandes étapes de la synthèse d'images:

- Description de la scène à visualiser

- Construction de la scène à l'aide d'outils de modélisation

- Création finale -"synthèse" finale- de l'image à partir des modèles réalisés

2.1. Description de la scène

De facon courante, les scènes tridimensionnelles sont décrites à l'aide de codes géométriques parmi lesquels nous pouvons citer:

- composition de FACES PLANES

- composition de SURFACES GAUCHES, approximées par des systèmes d'équations de type z=f(x,y), permettant un excellent réalisme -B-splines, fractales-

- représentations FIL DE FER - wire-frame -, chaque objet étant défini comme un ensemble d'arêtes réunissant ses sommets

- composition booléenne d'objets élémentaires tels que pyramides, cônes, parallèlépipèdes - ray casting par exemple -

2.2. Contruction de la scène

Cette phase consiste en la réalisation des modèles d'objets à partir des éléments fournis par la description. Deux types de modèlisation peuvent être évoqués dans cette phase:

- modèlisation de la visibilité de la scène

- modèlisation de l'aspect des objets

2.2.1. Modèlisation de la visibilité

Elle a pour but la détermination des objets ou portions d'objets visibles, en fonction de la structuration de la scène et des paramètres de prise de vue.

Les techniques mises en oeuvre sont connues sous le vocable d'algorithmes de visibilité ou d'élimination de parties cachées.

Nombre de procédés ont été développés au cours de ces dernières années, se heurtant tous à deux contraintes: la dépendance vis-à-vis du type de description d'une part, les performances liées à la complexité de la scène d'autre part.

Une étude complète à été réalisée sur ces problèmes par P.BOUKNE (BOU80) dans le cadre d'une thèse de Docteur-Ingénieur, où il a apporté

quelques améliorations et surtout recensé et classé les algorithmes existants.

Les critères de classification sont variés:

- nature des éléments composant la scène
- nature de la présentation: production de dessins au trait ou images
- nature des calculs: travail dans l'espace objet ou image
- méthode de base: par parcours du contour, divisions de la scène, balayage ligne par ligne, résolution point par point
- utilisation de propriétés topologiques de la scène, permettant une réduction du nombre d'éléments à traiter

Parmi les plus utilisés, citons les algorithmes de WATKINS (1970) -détermination de segments visibles par un balayage ligne-, ATHERTON et WEILER (1977) -détermination de portions de faces visibles-, NEWELL, NEWELL et SANCHA (1972) -détermination de faces visibles après tri et éventuellement découpage-, CATMULL (1974) ou algorithme dit du "Z-buffer" -travail en chaque point de l'image- ou encore SCHUMACKER (1969) -définition de priorités statiques et dynamiques-.

2.2.2. Modèlisation de l'aspect des objets

Cette phase a pour but la réalisation d'effets accroissant le réalisme de chacun des objets composant la scène.

Le lecteur trouvera dans un document de recherche de F.MARTINEZ (MAR79) le développement de ces éléments de modèlisation d'aspect.

Nous nous bornerons ici à énoncer la liste des attributs sur lesquels porte la modèlisation d'aspect:

- la coloration, et ses représentations spatiales en (Rouge, Vert, Bleu) ou (Teinte, Luminance, Saturation), et les textures
- la transparence, liée aux phénomènes physiques de réfraction et transmission
- l'éclairage, avec les techniques de calcul de luminance en un point et les méthodes d'optimisation pour l'objet global
- les ombres portées, problème parallèle à celui de visibilité et traité souvent simultanément -le point de vue étant remplacé par la position de la source lumineuse-

2.3. Synthèse finale de l'image

Cette phase finale construit l'image résultante des modèles de visibilité et d'aspect.

Elle met en oeuvre essentiellement deux techniques:

- remplissage de taches
- application de textures

2.3.1. Remplissage de taches

Cette opération consistant à générer le contour d'objet, et éventuellement le remplir dans le cas de surfaces, est traitée différemment selon le type de définition du contour.

* Définition mathématique: c'est le cas, entre autres, des vecteurs, cercles, ellipses, ... où des algorithmes typiques de numérisation sont appliqués, les plus connus étant ceux de BRESENHAM.
* Définition de contours polygonaux: si le contour est numérisé, la technique la plus fréquente est l'analyse de la mémoire de trame et la digitalisation du contour "codé" de manière à repèrer les points de croisement, segments horizontaux et points extrêmes du contour. Dans le cas d'un contour formel, on peut travailler soit par inversions successives, soit par décomposition en trapèzes.

2.3.2. Application des textures

L'intégration des modèles de textures dans la scène à visualiser pose deux types de problèmes:

- problèmes d'aliasing, consistant en une discontinuité au niveau de la texture d'un objet, résolus par des filtrages "anti-aliasing", lors de l'application elle-même des textures
- problèmes de même effet, mais dûs aux transformations géométriques subies par les objets

BLINN et NEWELL (1976) d'une part, CROWN (1977) d'autre part ont apporté des réponses à ces questions.

3. Les différentes technologies

Ce paragraphe est l'occasion de réaliser un rapide tour d'horizon des différentes technologies en matière de synthèse d'images, où nous présenterons tour à tour:

- les dispositifs d'affichage

- les dispositifs d'entretien
- les dispositifs de coloration
- des exemples de systèmes

3.1. Dispositifs d'affichage

La plupart d'entre eux font appel aux tubes à rayons cathodiques. Les nouvelles technologies telles que afficheurs fluorescents, plasma ou cristaux liquides sont encore très limitées.

Deux techniques sont utilisées par les dispositifs classiques:

- balayage cavalier: le faisceau d'électrons asservi en X et Y suit le contour du dessin à afficher. Très adapté au dessin au trait, excellente résolution, couleur limitée.
- balayage de trame: le faisceau d'électrons balaie l'écran ligne par ligne. Ses critères sont inverses de ceux du balayage cavalier.

3.2. Dispositifs d'entretien

Trois techniques sont présentes, les deux premières réservées au balayage cavalier, la troisième au balayage de trame.

- Tube mémoire: deux canons à basse énergie émettent un faisceau d'électrons secondaire d'entretien permettant la conservation de l'image
- Tube à liste de visualisation: un interpréteur graphique parcourt en permanence une mémoire, appelée liste de visualisation, contenant des instructions graphiques.
- Tube à mémoire de trame: la mémoire est ici une RAM contenant autant de points que l'image, et où chaque élément exprime la couleur du point de l'image en (R,V,B)

3.3. Dispositifs de coloration

Là encore deux techniques, selon le type de mode d'affichage :

- Tube à pénètration, pour ce qui est du balayage cavalier. L'écran est recouvert de deux couches de phosphore rouge et vert, donnant quatre couleurs au maximum -rouge,orange,jaune,vert- selon l'intensité de pénétration du faisceau d'électrons
- Tube à masque, pour ce qui est du balayage de trame. L'écran est tapissé de grains de phosphore organisés en triades, donnant les trois couleurs fondamentales Rouge,Vert et Bleu, chaque élément de la triade étant excité par un faisceau d'électrons filtré par un masque perforé.

Les techniques du tube 110 degrés PIL (Phase In Line) de THOMSON, ou TRINITRON de SONY sont des variantes de cette technique.

3.4. Exemples de systèmes

L'architecture interne des consoles de visualisation, dont les principales caractèristiques viennent d'être énoncées, permet de distinguer deux types de systèmes:

- systèmes minimaux
- systèmes généraux

3.4.1. Systèmes minimaux

La configuration minimale, représentée par la figure I.3 comporte outre un dispositif d'affichage, une mémoire d'entretien, un processeur d'entretien et un processeur d'échange, destiné à l'affectation et la consultation de la mémoire d'entretien.

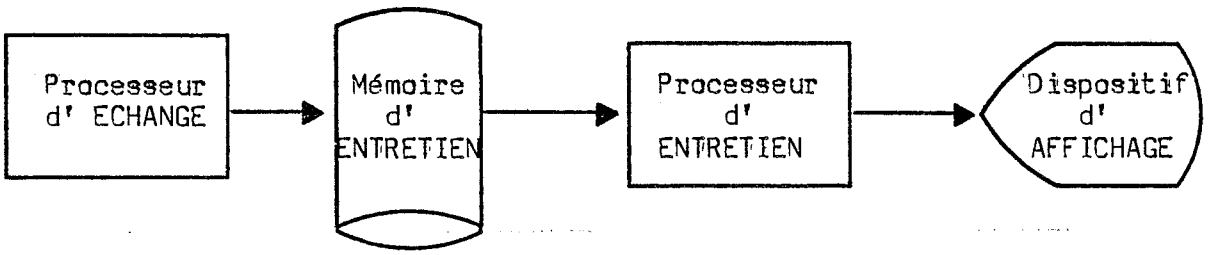


Figure I.3: Architecture minimale

Parmi ces systèmes, citons:

- le TEKTRONIX 4010 à tube mémoire
- les terminaux SECAPA à balayage de trame, munis de coprocesseurs graphiques de type EFCIS EF9365/EF9366
- les IBM 2250, 3250 à liste de visualisation

3.4.2. Systèmes généraux

Ils se caractèrisent par une tendance à des processus plus évolués, par l'adjonction de microprocesseurs, ou microordinateurs et de logiciel spécialisé.

Un exemple d'un tel système est donné en figure I.4 avec le RAMTEK 9400, qui associe les avantages du balayage cavalier -définition de l'image par vecteurs- et de trame -remplissage de taches,couleur-.

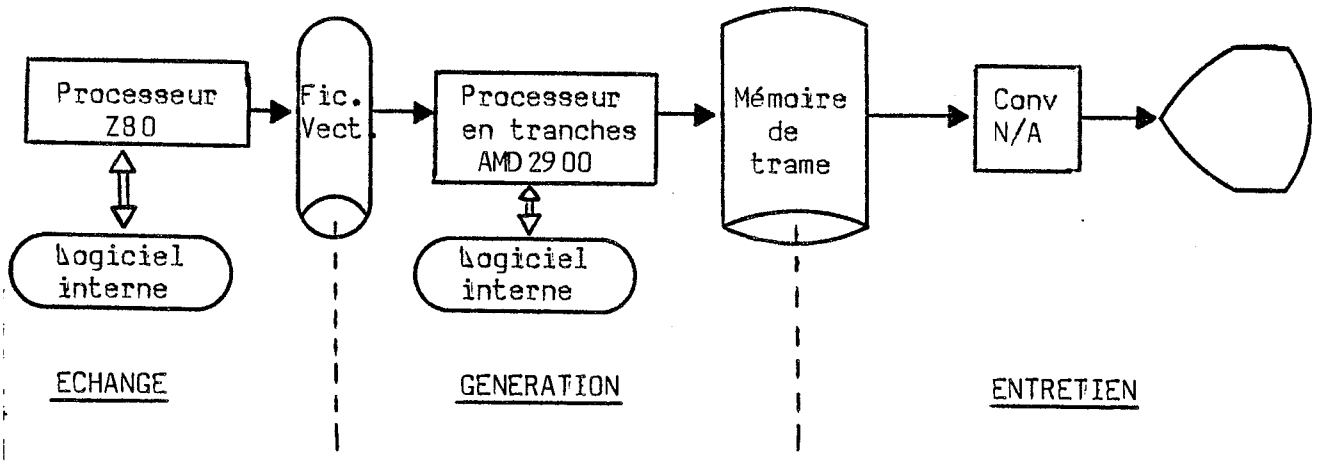


Figure I.4: RAMTEK 9400

Terminons ce paragraphe, et par là même ce chapitre par l'évocation de systèmes spécifiques dédiés à un processus particulier. Citons l'exemple du système CVD/2 de RIVERS COMPUTER CORPORATION permettant l'animation d'images en temps réel.

CHAPITRE II : ORGANISATION D'UN SYSTEME DE SYNTHESE D'IMAGES - ASPECTS CONCEPTEUR ET TECHNIQUE

Nous nous attacherons dans ce chapitre à présenter d'une part les éléments de base conceptuels décrivant de facon succinte opérations et processus rencontrés dans un système de synthèse d'images, d'autre part à proposer une organisation d'un tel système capable :

- d'assurer l'indépendance vis-à-vis du matériel
- de satisfaire le plus grand nombre d'applications
- de permettre une grande diversité de performances

Ces différents points ont été l'objet d'une étude de F.MARTINEZ dans le cadre d'une thèse de Docteur d'Etat (MAR82).

1. Elements de base

Nous avons évoqué au chapitre précédent -cf. paragraphe 2- les trois étapes de la synthèse d'images. Voyons maintenant les principales opérations mises en oeuvre au cours de ces différentes phases.

1.1. Les opérations

1.1.1. Description de la maquette

Cette étape nécessite la description puis la modèlisation de toutes les classes d'attributs :

- description morphologique (M) de tel ou tel objet
- situation des objets les uns par rapport aux autres (G)
- affecter aux objets un aspect (A), indiquer les conditions d'éclairage (E)
- identifier (I) et structurer (S) les objets

1.1.2. Construction de la maquette

Interviennent ici essentiellement des opérations de génération permettant l'obtention d'une information complète, à l'aide d'algorithmes, d'opérateurs de composition, et en faisant référence à des banques d'informations. Il est à noter que le temps passé dans cette phase est important.

1.1.3. Prise de vue

Elle s'applique à partir de paramètres spécifiques -point de vue, direction de visée, format de vue et domaine visible- et met en jeu des attributs géométriques que nous noterons (Gv). Ils expriment la transformation des coordonnées exprimées dans le repère de la maquette en coordonnées exprimées dans le repère de la vue.

1.1.4. Affichage

Il s'agit de cadrage de la vue pour l'obtention de l'image finale. Les attributs géométriques intervenant sont notés (Ga) et permettent la transformation des coordonnées exprimées dans le repère de la vue en coordonnées exprimées dans celui de l'image, tous deux bidimensionnels.

1.1.5. Récapitulatif

Nous pouvons donner ici une liste d'opérations entrant dans un système de synthèse d'images :

- Description-I, Description-M, Description-A, Description-G
- Description-E, Description-S
- Description-Gv
- Description-Ga
- Visualisation

Ces opérations s'inscrivent dans un contexte d'échange, que nous abordons ci-après à travers les différents processus d'un système de synthèse d' images.

1.2. Les processus

En effet, un système de synthèse d'images joue le rôle d'intermédiaire entre deux interlocuteurs qui sont, d'une part le programme d'application, d'autre part l'opérateur, ce à travers quatre processus, schématisés sur la figure II.1.

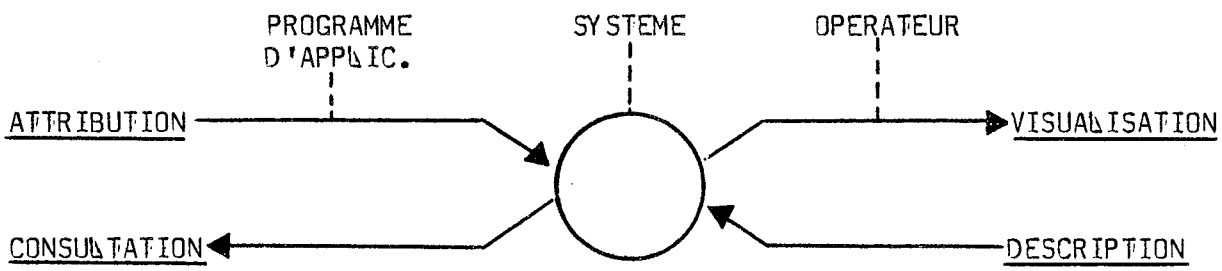


Figure II.1: Les 4 processus d'un système de synthèse d'images

Ces quatre processus, sous le contrôle de l'application, peuvent être invoqués par le biais de primitives. L'objet des paragraphes suivants est de définir un peu mieux ces processus, et de leur associer des primitives possibles.

1.2.1. L'attribution

Il s'agit d'une affectation par le programme d'application d'attributs dans les structures du système :

ATTRIBUER (id-elts, type-att, info)

id-elts: identité du, ou des élément(s) à laquelle est affectée l'information
type-att: type de l'attribut
info: valeur de l'information

1.2.2. La consultation

Elle concerne la récupération des attributs du système par le programme :

CONSULTER (id-elts, type-att, info)

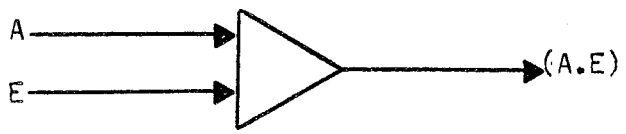
info: est ici le résultat du processus

1.2.3. La visualisation

Un processus de visualisation doit assurer la synthèse des attributs descriptifs des éléments en une information finale exprimant la couleur en chaque point de l'image. De facon générale, il se décompose en quatre opérateurs élémentaires :

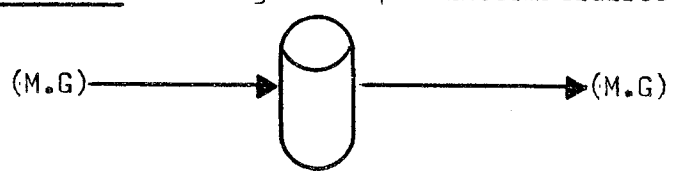
* de synthèse: de deux attributs de base

ex:



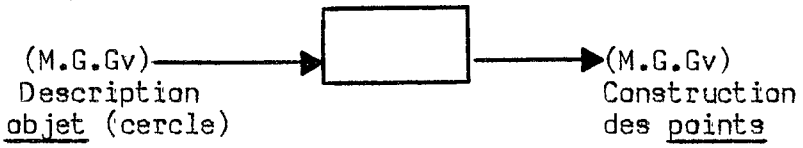
* de mémorisation: stockage d'étapes intermédiaires

ex:



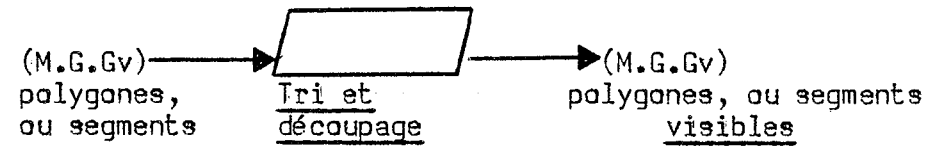
* de construction: transformation d'un type d'attribut dans un autre

ex:



* de composition: d'attributs de même type de plusieurs éléments

ex:



La primitive associée est :
VISUALISER (id-elts, processus)

processus: détermine l'ordre et la nature des opérateurs

1.2.4. La description

Il s'agit de décrire les informations de base de la maquette (I,M,A,G,E,S,Gv,Ga) à l'aide de données fournies par l'utilisateur. Cela est possible par transformation d'informations initiales qui sont :

- des coordonnées obtenues par des dispositifs de saisie

- des paramètres numériques ou alphanumériques caractèrisant les attributs

Il est à noter que certaines opérations de construction pouvant être effectuées lors de cette phase de description -afin par exemple d'éviter une perte de temps au moment de la visualisation-, la frontière entre description et visualisation n'est pas toujours évidente.

Ainsi, dans le cas particulier où le processus de description s'effectue par référence à des élements déjà définis, impliquant une identification par désignation. Une pseudo-visualisation est alors nécessaire, regroupant les opérateurs inverses de ceux de synthèse, et ce jusqu'à la mémorisation d'attributs concernant l'élément identifié, afin de vérifier les attributs du type identifié.

La primitive associée à ce processus de description est la suivante:

DECRIRE (id-elts, processus, type-att)

2. Organisation générale du système

2.1. La notion de synthétiseur

Est appelée synthètiseur une unité de traitement possédant les caractèristiques suivantes:

- présence d'un processeur au moins, qu'il soit cablé, micro-programmé, ou programmé
- présence d'une ressource locale de mémoire au moins
- existence d'au moins un processus de visualisation et un processus d'attribution dans cette mémoire
- possibilité de processus de description et consultation

Un système de synthèse d'images peut alors être défini comme une suite de synthétiseurs, dont le dernier est le tube cathodique, représentant ainsi des couches de synthétiseurs.

Dans le paragraphe suivant, sont abordées les configurations matérielles les plus représentatives et leur influence sur l'organisation générale du système.

2.2. Influence de la configuration matérielle

Nous évoquerons ici trois situations caractéristiques, différenciées selon les critères suivants :

- nombre de synthétiseurs indépendants, ainsi que leur type
- capacité et type de mémoires locales associées à chacun des synthétiseurs
- éléments et processus acceptés par chaque synthétiseur

2.2.1. Console bas de gamme

Cette configuration minimale se compose de deux synthétiseurs :

- calculateur hôte avec son logiciel et ses ressources de mémorisation
- console de visualisation "bas de gamme"

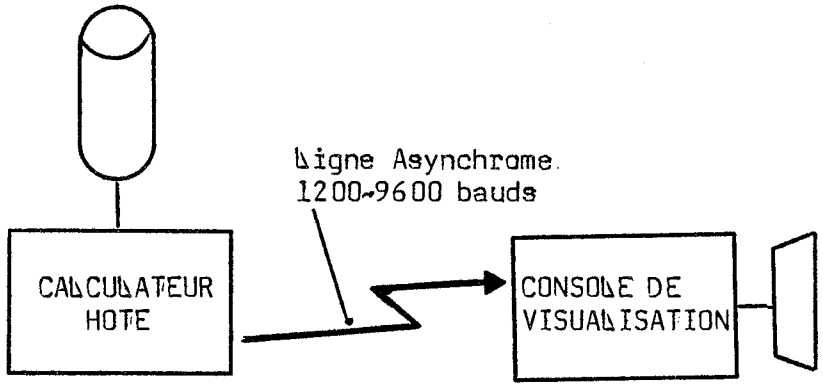


Figure II.2: Console bas de gamme

Les éléments proposés sont assez pauvres : point, vecteur, caractères, ainsi que les processus disponibles limités à l'attribution et la visualisation de ces éléments, assurés essentiellement par le logiciel implanté au niveau du calculateur hôte.

2.2.2. Console évoluée

La configuration est la suivante, se caractérisant par l'adjonction d'un synthétiseur intermédiaire avec son logiciel et ses propres ressources de mémorisation, permettant donc des modifications de l'image sans intervention du calculateur principal.

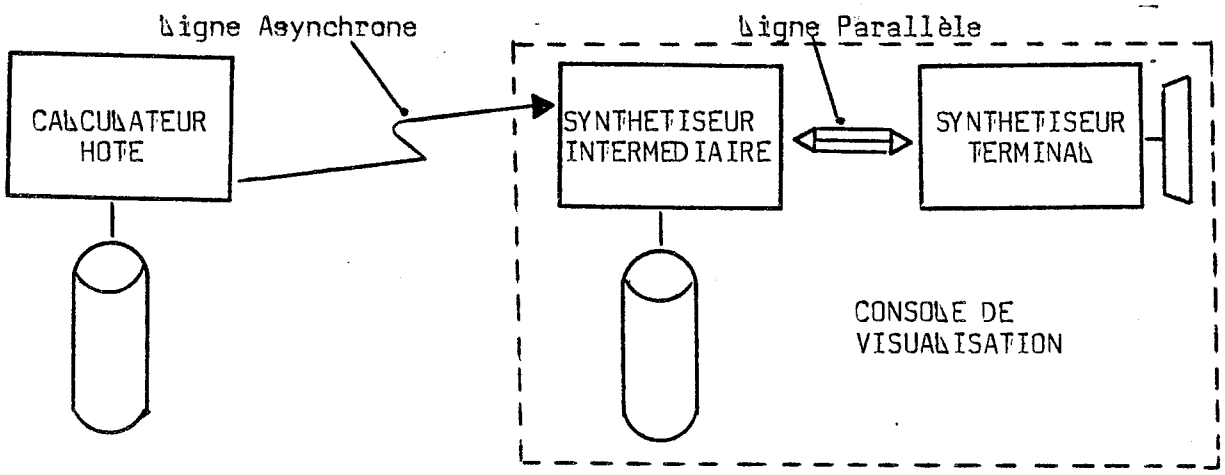


Figure II.3: Console évoluée

Cette organisation qui tend à se généraliser de par l'utilisation des microprocesseurs pose des problèmes lors de la mise en oeuvre d'un tel type de système. Il y a d'une part risque d'incohérence si le logiciel ne détecte pas les lacunes du matériel et n'offre pas de solution de remplacement. D'autre part, une certaine dissymétrie est à craindre de par l'impossibilité à décrire et consulter les mêmes types d'attributs que ceux acceptés en entrée. Ceci implique l'application d'opérations d'analyse pour restituer ces attributs initiaux ou encore leur mémorisation, et donc duplication au niveau logiciel.

2.2.3. Connexion à un calculateur satellite

Dans cette situation, la console de visualisation est reliée au calculateur principal via un calculateur intermédiaire possédant sa mémoire propre. Le calculateur satellite, comme le schématise la figure II.4, représente une couche supplémentaire, cohérente et symétrique, permettant au calculateur hôte d'asservir le terminal aux exigences de l'application.

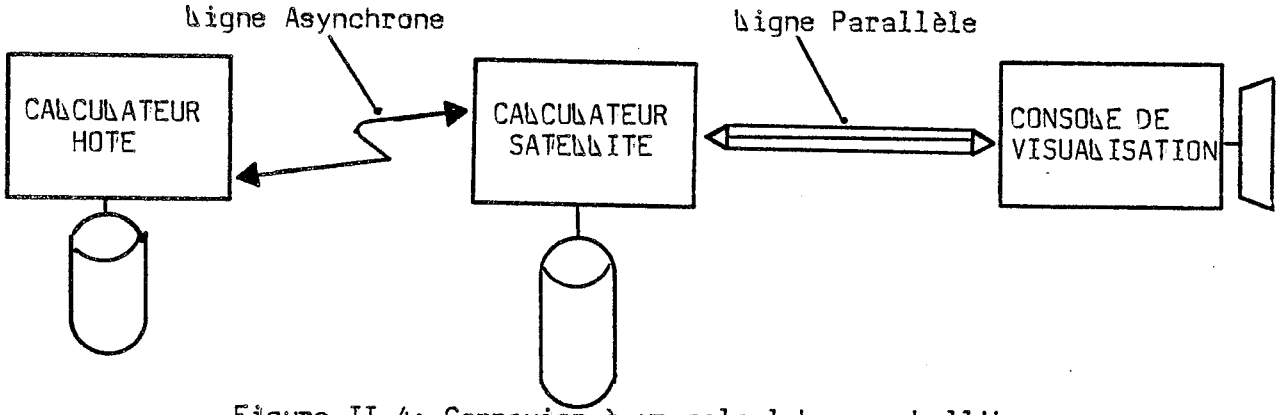


Figure II.4: Connexion à un calculateur satellite

2.2.4. Configuration multiple

Il est en fait fréquent qu'un site soit exposé aux trois situations présentées ci-dessus. Il est alors intéressant que le calculateur satellite soit placé en frontal de tous les terminaux, simulant ainsi la couche unique à laquelle s'adresse le calculateur hôte et permettant dans les meilleures conditions l'indépendance du logiciel.

Le découpage en synthétiseurs réalisé, il reste à organiser et réaliser chacun d'eux. La réalisation peut revêtir de nombreux aspects -logiciel sur un calculateur standard, microprocesseur standard, microprocesseurs spécialement adaptés et intégrés dans des boitiers VLSI, logique cablée-. Quant à l'organisation interne de chacun des synthétiseurs, elle dépend étroitement du ou des processus à réaliser, et c'est ce point qu'aborde le paragraphe suivant.

2.3. Organisation interne d'un synthétiseur

2.3.1. Cas particulier: synthètiseur monoprocessus

C'est le cas le plus simple, un tel système ne comportant qu'un seul type d'élément à synthétiser sur un seul synthétiseur "terminal". Il découle de cette situation qu'il n'y a qu'un seul type de processus -de visualisation , de description, d'attribution, de consultation-. L'amélioration de performances passe, d'une part par la recherche de techniques optimisant le processus -et tout particulièrement celui de visualisation, le plus délicat et le plus complexe-, d'autre part par l'organisation du synthétiseur en fonction du processus choisi.

* Ordonnancement du processus: optimiser celui-ci revient à optimiser l'influence de certains attributs privilégiés. Par ce terme est désigné le nombre d'opérations élémentaires -cf. paragraphe 1.2.3. de ce chapitre- à effectuer pour visualiser l'élement concerné. Deux techniques sont utilisables:

- mémorisation des attributs synthétisés, diminuant l'influence des attributs entrant en jeu après cette mémorisation, utile pour améliorer par exemple le calcul de plusieurs points de vue ou l'identification par désignation.

- pénétration des attributs non synthétisés, consistant à modifier le processus afin que l'attribut à privilégier se retrouve le plus près possible du dernier opérateur du processus.

La conjugaison de ces techniques permet d'envisager un grand nombre de solutions, la situation se compliquant davantage encore lorsque plusieurs éléments uniques doivent être synthétisés simultanément.

* Architecture du synthétiseur: elle dépend essentiellement du nombre et de l'architecture des processeurs utilisés ainsi que des tâches allouées à chacun d'eux, le cas minimal étant la charge de toutes les opérations du processus par un processeur unique, le cas maximal étant l'affectation d'un processeur à chaque opérateur. Enonçons donc en fonction de ces critères les divers compromis possibles en ce qui concerne l'architecture:

- séquentiel intégral: 1 processeur effectue 1 opération sur 1 élément unique, à un instant donné

- pipe-line: chaque sous-processus est réalisé par un processeur, la synchronisation de chacune des phases s'opérant soit automatiquement si les sous-processus sont de période égale, soit par l'adjonction de mémoires- tampon dans le cas contraire.

- parallèlisme intra-processus: entre différents opèrateurs portant sur des attributs différents

- parallèlisme inter-processus: il permet de traiter simultanément tous les éléments de la maquette

Examinons à présent le cas le plus général, à savoir celui des synthétiseurs multiprocessus, les problèmes posés ainsi que les organisations possibles.

2.3.2. Organisation d'un synthétiseur multiprocessus

* les problèmes liés aux processus multiples sont de deux ordres:

- synchronisation entre processus différents: cette difficulté est amplifiée par la différence des éléments traités par les différents processus, et l' impossibilité de trouver des opérateurs capables de traiter des éléments quelconques. Il s'agit alors de déterminer le point de convergence, et ce :
  . par suppression des disparités, en transformant les èléments initiaux disparates en un élément standard, solution difficilement conciliable avec le réalisme des images
  . par convergence au niveau du pixel, celui-ci étant par principe l'élément terminal de tout processus de synthèse
  . par convergence intermèdiaire, compromis lié à la possibilité de partitionner les processus en sous-processus terminaux identiques, et dépendant donc fortement des types d'éléments traités.

- réduction du nombre d'opérateurs: celui-ci est très important pour un synthétiseur général capable de répondre au maximum de situations. Essayons d'effectuer une approche combinatoire de ce nombre d'opérateurs, noté T.

Un synthétiseur défini comme ci-dessus doit considérer les classes d' applications à satisfaire C -NC classes- et les types de matériels possibles M -NM matériels-. La figure II.5 représente cette situation, U(C,M) représentant l'unité du synthétiseur permettant de traiter l'application C avec un matériel M.

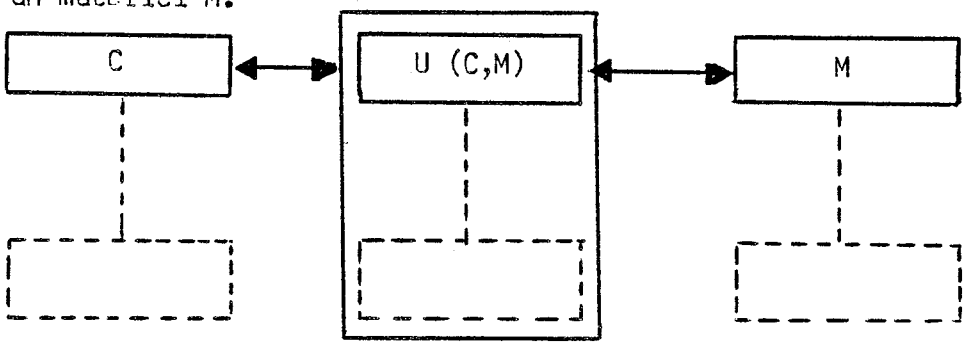


Figure II.5: Synthétiseur adapté à chaque situation

T est alors donné par:
T = NC * NM * n * k
si n est le nombre moyen de processus par classe
et k le nombre moyen d'opérateurs par processus.

Deux méthodes d'organisation permettent de réduire T.

* Organisation hiérarchique: elle repose sur la séparation entre l'ordonnancement du processus d'une part, les opérateurs utilisés d'autre part. Ces opérateurs peuvent être scindés en deux groupes:

- opérateurs de synthèse, construction et composition, opérateurs de base à la description et la visualisation

- opérateurs de mémorisation, nécessaires aux deux processus ci-dessus cités, mais nécessitant des structures de données importantes dont la gestion doit être assurée.

Ainsi se dessinent trois unités dans ce type d'organisation, comme le représente la figure II.6:

- unité de contrôle: outre son rôle prédominant d'ordonnancement des processus, elle assure les tâches suivantes:
  . banque de processus vis-à-vis des synthétiseurs "amont"
  . liaison avec le synthétiseur "aval", en invoquant ses primitives de base
  . indépendance avec l'application en proposant un jeu de primitives standard -cf. paragraphe 1.2. de ce chapitre-
  . transfert d'informations entre les deux autres unités
  . contrôle du déroulement des processus

- unité de description-visualisation: c'est une banque d'opérateurs -synthèse, construction, composition- assurant l'indépendance vis-à-vis des dispositifs de saisie et d'affichage

- unité de communication: elle assure la gestion des structures de données nécessaires aux opérateurs de mémorisation. La solution proposée quant à cette structure est une base de données unique, banalisée permettant la structuration, l'affectation et la recherche d'informations graphiques.

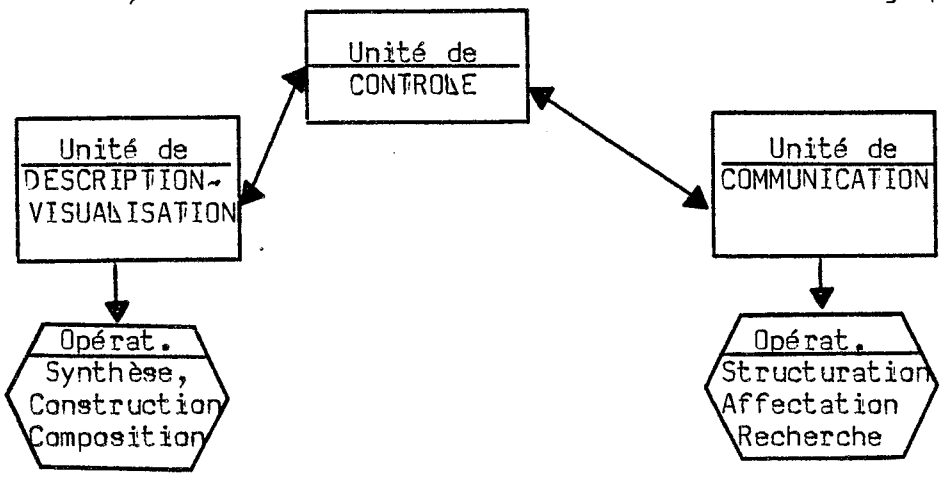


Figure II.6: Organisation hiérarchique

Les deux problèmes auxquels se heurte une telle organisation -dont la figure II.7 donne un exemple appliqué aux différents synthétiseurs logiciel et matériel d'un système de synthèse- sont d'une part la réalisation de la base de données banalisée répondant ainsi aux diverses structures des synthétiseurs, d'autre part la prise en charge et la gestion des processus selon les types d'attributs et le matériel utilisé.

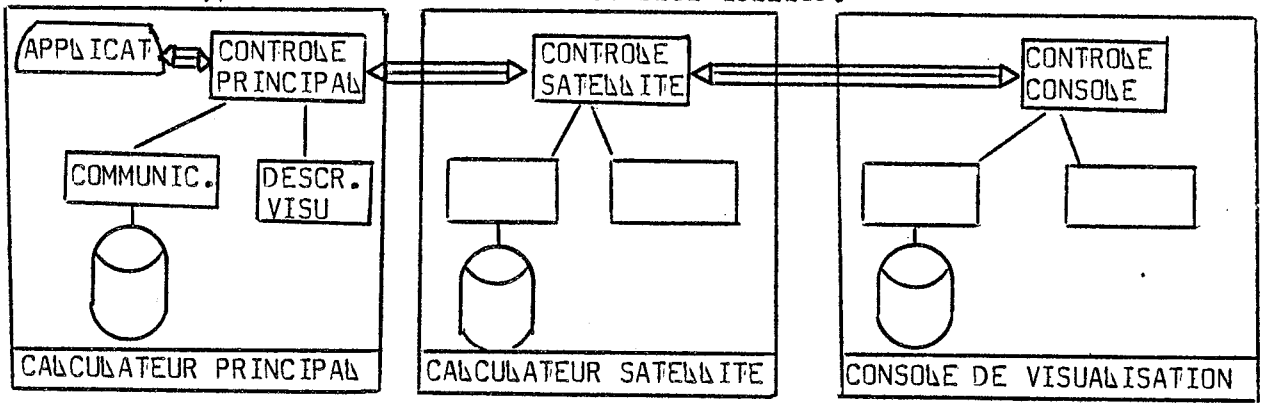


Figure II.7: Exemple d'organisation hiérarchique

* Partitionnement en couches: cette organisation, la plus courante dans les systèmes graphiques actuels, se caractèrise par le découpage du synthétiseur en unités de traitement -ou couches- simulant autant de synthétiseurs complets pour les éléments de leur univers. Chaque unité comporte donc sa propre structure de données ainsi que l'ensemble des processus énoncés au cours des paragraphes précédents.

Le partitionnement consiste en général à constituer deux couches indépendantes, par l'adjonction d'un univers intermèdiaire S, assez "standard" pour convenir au maximum de situations -applications et matériels-. Si l'on reprend la terminologie employée dans le paragraphe 2.3.2., chaque unité U(C,M) devient U(C,S) et U(S,M), l'une dépendante de l'application et l'autre du matériel. Le principal avantage de cette organisation est donc la double indépendance ainsi réalisée. Quant aux limitations, elles découlent directement de la multiplication des couches:

- augmentation en temps et place mémoire
- gestion des informations de structure compliquée
- propriétés du système liées à la couche la moins performante

C'est sur cet aspect performances que nous terminerons ce chapitre, après avoir représenté par la figure II.8 un exemple de partitionnement en couches et son aspect "inflationniste" en particulier au niveau de la transmission des informations.

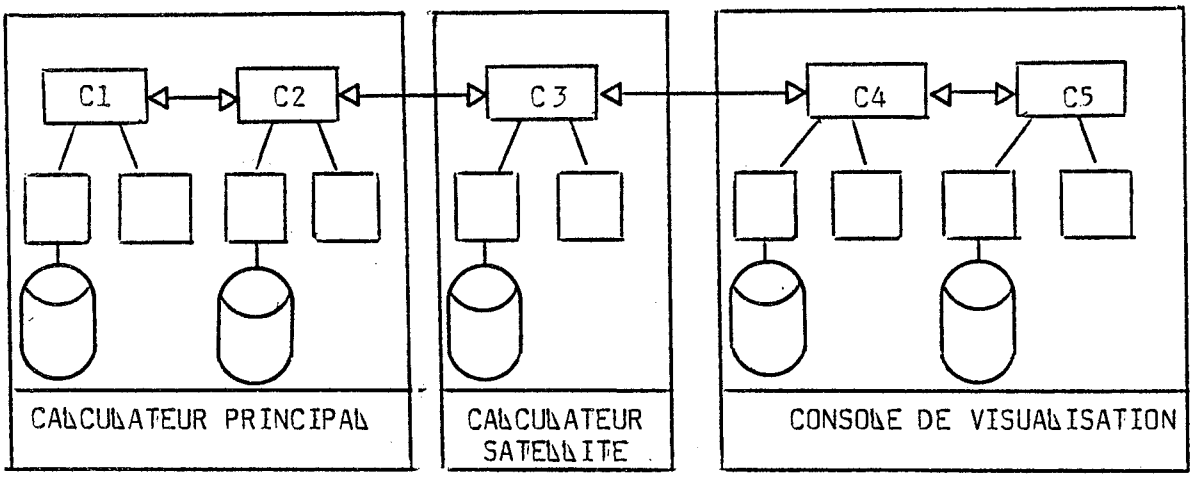


Figure II.8: Exemple de partitionnement en couches

3. Les performances d'un système

Au vu des idées énoncées précédemment, il est clair que la conception et la réalisation d'un système de synthèse d'images passe par l'étude des objectifs à atteindre. Parmi ceux-ci, citons:

- les performances intrinsèques du système: il s'agit d'un compromis entre trois éléments que sont la qualité d'image, le temps de réponse et la complexité de la scène.

- la puissance d'interaction en matière de description s'appuyant sur deux opérations de base: identification par désignation et collecte d'informations

- l'indépendance et adaptativité, critères permettant de différencier systèmes généraux et spécifiques

- le coût du développement

Les différentes notions théoriques que nous venons d'énoncer forment la "charpente" du système de synthèse d'images élaboré dans le groupe "Image" du laboratoire ARTEMIS de l'IMAG. C'est ce système que nous nous proposons de présenter dans le chapitre suivant.

CHAPITRE III : UN EXEMPLE DE SYSTEME - ASPECTS MATERIELS ET LOGICIELS

Ce chapitre présente un exemple de système de synthèse d'images, conçu et réalisé conformément aux concepts présentés au chapitre précédent. Un point d'ensemble sur ce système a été donné par F.MARTINEZ dans le cadre de sa thèse d'Etat (MAR82).

Ce système multiprocessus décharge au maximum l'application tant en ce qui concerne la modèlisation et la structuration propres à la représentation graphique des objets que ce qui touche l'éxécution des processus , ne se contentant pas de présenter une simple bibliothèque d'actions dont la gestion serait assurée par l'application.

Nous évoquerons successivement :

- la présentation générale du système
- les synthétiseurs cablés
- les synthétiseurs programmables

1. Présentation générale du système

Le souci premier de ce système étant de répondre au maximum de situations, une grande diversité a été recherchée en ce qui concerne:

- les applications: en premier lieu CAO, mais aussi EAO, domaine artistique, cadre scientifique
- les technologies d'affichage: balayage cavalier pour le dessin au trait, balayage de trame pour les images réalistes
- les configurations: terminaux bas de gamme, évolué, calculateur satellite
- les architectures: pipe-line, et parallèle
- les moyens d'entrée: réticule, tablette à numériser

En ce qui concerne le domaine des images réalistes, choix a été fait de réaliser un terminal interactif pour la synthèse de telles images, privilégiant en particulier les opérations cablées "après la mémoire de trame". Il s'agit du prototype HELIOS dont nous verrons le fonctionnement dans le paragraphe 2 de ce chapitre, et dont on peut trouver une étude détaillée dans une thèse de Docteur-Ingénieur de F.NUNES FERREIRA (FER81).

De ces considérations découle la configuration matérielle suivante,

illustrée pas la figure III.1:

- un calculateur principal: successivement CII-HB 68 du C.I.C.G. sous MULTICS, VAX 11/780 de MICADO sous VMS
- le poste de travail HELIOS
- des terminaux graphiques classiques, en particulier ceux compatibles avec le TEKTRONIX 4010
- un traceur BENSON connecté au CII-HB

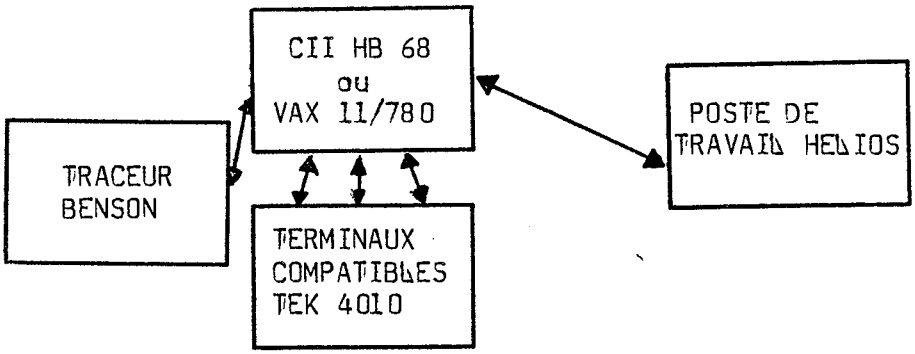


Figure III.1: Configuration matérielle

Il est à noter que le poste de travail HELIOS comporte deux synthétiseurs, l'un cablé, l'autre programmé sur calculateur satellite pilotant le premier.

Ainsi deux types de configurations peuvent se présenter, ainsi que le représente la figure III.2:

- une configuration avec deux couches de synthétiseurs: calculateur principal, terminaux 4010 ou traceur
- une configuration avec trois couches: calculateur principal, calculateur pilote, synthétiseur cablé

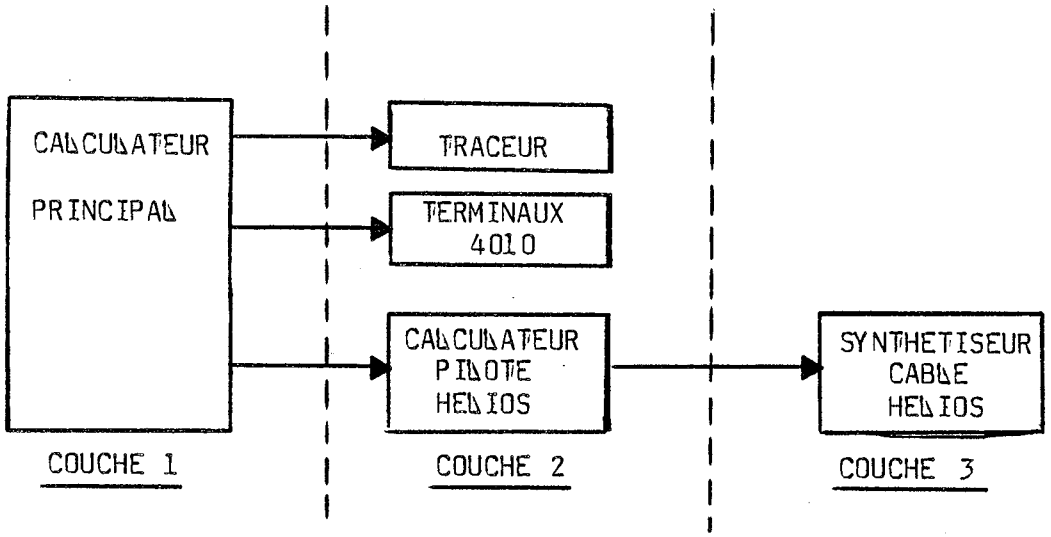


Figure III.2: Les différentes configurations

Nous terminerons cette présentation par l'organisation générale du système. Chaque couche de synthétiseur a une organisation hiérarchique, dont la schématisation donnée par la figure II.7 du paragraphe 2.3.2. du chapitre II peut très bien s'appliquer ici.

2. Les synthétiseurs cablés : le prototype HELIOS

Ce terminal, orienté vers la synthèse d'images réalistes, présente les possibilités suivantes:

- haut degré d'interactivité
- modification en temps réel de l'aspect des objets, des attributs d'éclairage et quelques attributs géométriques
- identification instantanée par désignation sur l'écran à l'aide d'un réticule intégré
- calcul en temps réel de la projection des textures et des réflexions diffuse et spéculaire

Nous présenterons successivement dans ce paragraphe la configuration matérielle d'HELIOS, celle-là même du premier prototype réalisé en 1981, puis son architecture générale, une notion primordiale et originale en ce qui concerne la visualisation -la face plane- et enfin un énoncé résumant les fonctions des différents processeurs.

Nous terminerons par l'évocation des versions suivantes d'HELIOS et des améliorations ainsi apportées.

2.1. La configuration matérielle d'HELIOS

Elle regroupe, comme nous l'avons annoncé dans le paragraphe précédent, deux synthétiseurs:

- l'un cablé
- l'autre programmé sur un calculateur satellite et pilotant le premier

La figure III.3 schématise cette configuration.

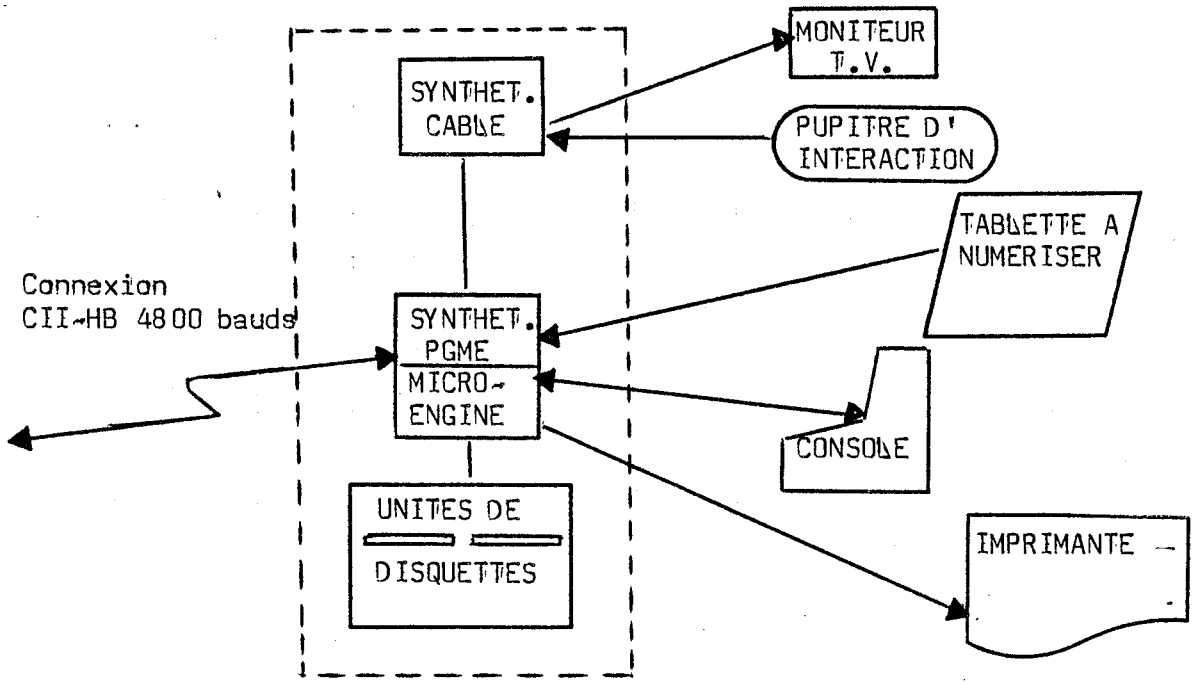


Figure III.3: Configuration d'HELIOS

Les différents éléments de cette configuration, sur laquelle nous reviendrons dans le chapitre V, sont les suivants:

- les 13 cartes du synthétiseur cablé
- le calculateur pilote MICROENGINE de WESTERN DIGITAL comportant le système UCSD

- la double unité de disquettes REMEX 40 (double face, double densité) d'une capacité de 2M octets

- le moniteur SONY à tube Trinitron

- le pupitre d'interaction permettant le contrôle du réticule

- la tablette à numériser SORED, d'une dimension de 50cm x 50cm, et d'une résolution de 0,025mm

- la console opérateur RETROGRAPHICS VT640 de DEC, permettant des fonctions de dessin au trait compatibles avec celles des terminaux de la série TEK 4010

- l'imprimante DIABLO 1641

2.2. L'architecture générale

2.2.1. Organisation hiérarchique

Elle s'articule donc autour de trois unités :

- unité de contrôle: un processeur cablé assure les échanges avec le calculateur pilote et l'ordonnancement des opérateurs -génération des signaux de synchronisation nécessaires aux autres processus et au moniteur TV-

- unité de communication: un processeur cablé gère les processus d'attribution et de consultation des différentes mémoires concernées

- unité de description-visualisation: quatre processeurs cablés assurent le processus de visualisation:

  * calcul de visibilité

  * calcul des textures

  * calcul de la réflexion

  * calcul de l'éclairage et synthèse finale

Un processeur supplémentaire permet le processus de description intégré à l'aide du réticule.

2.2.2. Les structures de données

Les informations traitées par ces processus sont enregistrées dans six mémoires par le logiciel pilote:

* mémoire de trame, ou plans d'identification

* table des faces

* banque des textures

* banque des modèles de réflexion

* paramètres d'éclairage

* paramètres de fenêtrage

2.3. Originalité d'HELIOS : la notion de face plane

C'est l'élément unique manipulé par le terminal, défini comme un ensemble homogène de points, c'est-à-dire possédant la même identité, le même aspect et les mêmes attributs géométriques.

Les attributs d'aspect sont:

- la texture, représentant la distribution des couleurs du matériau constituant la face

- la réflexion qui exprime la faculté du matériau à réfléchir la lumière

- la visibilité ou invisibilité, la transparence ou opacité

Les attributs géométriques représentent le repère tridimensionnel (U,V,N) attaché à la face, exprimé dans le repère de l'écran (x,y,z). (U,V) sont les vecteurs de base du plan de la face, N le vecteur normal à ce plan - cf. figure III.4-.

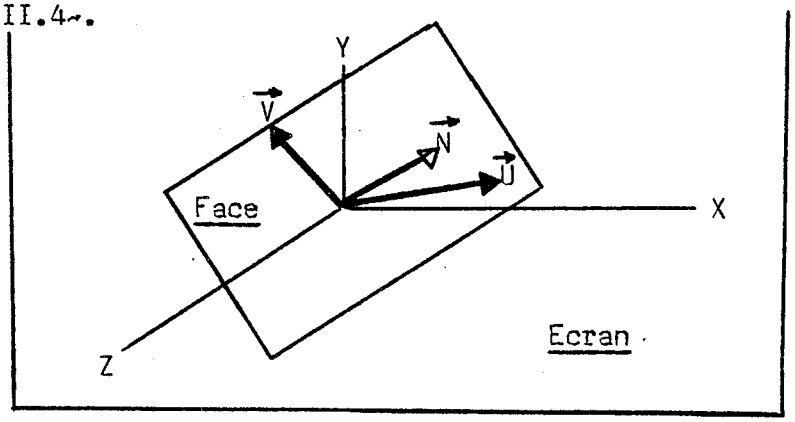


Figure III.4: Repère associé à une face

Enfin l'identité d'une face est constituée d'un numéro associé à celle-ci, compris entre 0 et 1023, le nombre maximum de faces ayant été fixé à 1024. Ce dernier peut sembler un peu limitatif, mais le nombre réel de faces composant une scène peut être bien plus important, puisqu'une "face Helios" peut représenter n faces qui seraient situées dans des plans parallèles et constituées du même matériau.

2.4. Les différents processeurs

Avant d'évoquer les fonctions de ceux-ci, nous indiquerons les structures de données associées.

2.4.1. Processeur de visibilité

* plans d'identification: HELIOS permet de gèrer 8 plans d'identification, chacun étant une mémoire de trame de 512x512 points de 10 bits, contenant le numéro d'identification des faces à visualiser (0 à 1023).
Le dernier plan, appelé plan de fond, ne mémorise qu'un numéro de face visible en tout point de l'écran, si une autre face n'a pas été attribuée.

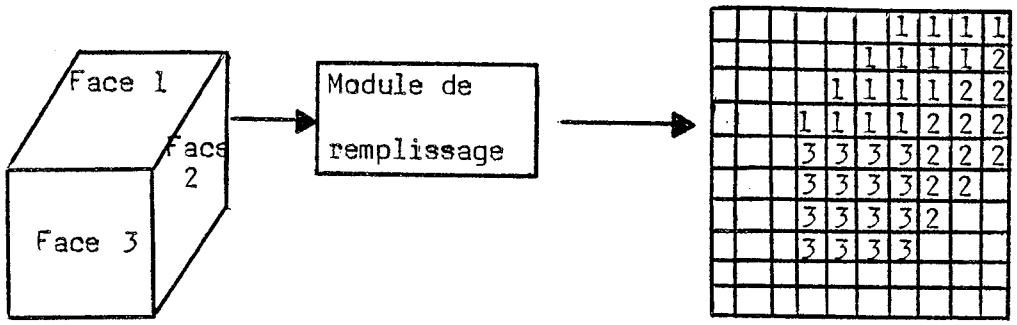


Figure III.5: Schématisation du plan d'identification

* La fonction fenêtre: cette fonction, existant pour les 8 plans, permet de décrire la portion carrée de la vue à afficher sur l'écran, à l'aide des paramètres suivants:

- coin supérieur gauche: déplacement en x,y de 0 à 511
- taille: 512x512, 256x256, 128x128, 64x64
- mode de découpage: visibilité ou invisibilité hors fenêtre

La figure III.6 montre un exemple de fenêtre.

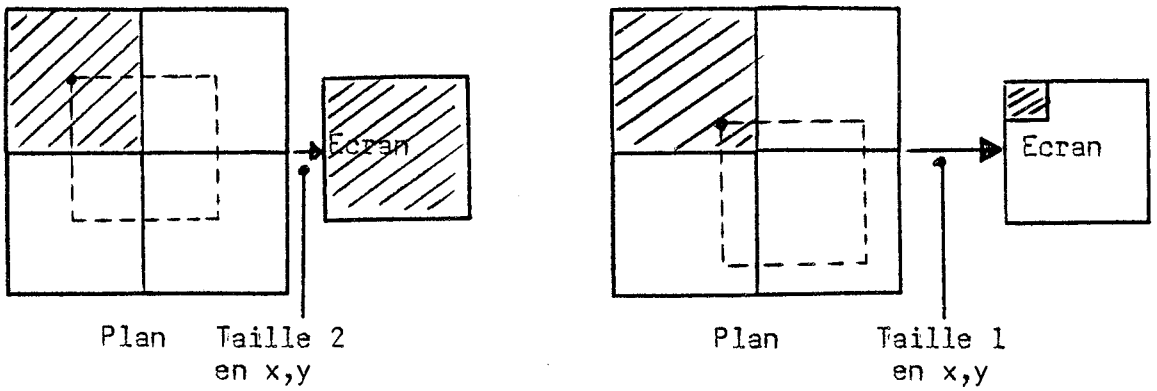


Figure III.6: Fenêtre avec mode de découpage invisible

* Etude de visibilité: détermine la face visible en chaque point de l'image, à partir de:

- priorité des plans d'identification
- mode de découpage de la fenêtre
- visibilité ou transparence des faces

Le plan 0 possède la plus forte priorité, le plan de fond la plus faible.

En ce qui concerne la transparence, seul le plan 0 accepte les faces munies de cet indicateur.

2.4.2. Processeur des textures

* La notion de texture: se caractèrise par les trois points suivants:

- chaque point exprime la couleur intrinsèque de la face
- chaque texture est supposée définie dans le plan de la face
- des textures de tailles différentes (16x16 à 512x512 points) peuvent être utilisées simultanément, ce par le biais d'une mémoire de textures

Les couleurs sont modélisées en trichromie rouge, vert, bleu sur 12 bits:

. Bits 11-8 : Vert
. 7-4 : Rouge
. 3-0 : Bleu

* La banque des textures: l'originalité de celle-ci réside dans le fait que l'adresse de la texture donne implicitement la taille, adresse du coin supérieur gauche et exprimée sur 10 bits (5 pour l'abscisse, 5 pour l'ordonnée).

Le nombre maximum de textures de même taille est:

- 1 : 512x512
- 3 : 256x256
- 12 : 128x128
- 48 : 64x64
- 192 : 32x32

* La projection des textures: seules les projections axonométriques -point de vue à l'infini- sont envisagées afin de simplifier le calcul cablé. Les textures étant définies dans le plan de la face, il s'agit de déterminer pour chaque point (xe,ye) de la vue, le point (xf,yf) correspondant dans le plan de la face. Cette transformation est obtenue par une matrice de changement de base M précalculée par logiciel et enregistrée dans la table des faces.

$$\begin{vmatrix} xf \\ yf \end{vmatrix} = M \times \begin{vmatrix} xe \\ ye \end{vmatrix}$$

* Le pavage des textures: il est effectué par matériel, afin d'éviter des problèmes d'aliasing, comme le montre la figure III.7.

Figure III.7: Pavage des textures

Après obtention du point (xf,yf), il faut calculer le point résultant dans la mémoire de textures -coordonnées (xb,yb)-, ce grâce au pointeur de textures (xa,ya) enregistré dans la table des faces:

$$\begin{vmatrix} xb \\ yb \end{vmatrix} = \begin{vmatrix} xa \\ ya \end{vmatrix} \times \text{taille texture} + \begin{vmatrix} xf \\ yf \end{vmatrix} \times \text{modulo taille}$$

2.4.3. Processeur de réflexion

Il détermine les coefficients de réflexion diffuse et spéculaire en chaque point de l'écran. Ils sont donnés par les formules suivantes:

$Rd = D(l)$
$Rs = S(m)$

Rd,Rs:coefficients de réflexion diffuse et spéculaire
D,S :modèles de réflexion diffuse et spéculaire
l :angle entre N et L
m :angle entre N et M
N :normale à la face
L :direction source lumineuse

M :direction médiane entre source et observateur

La figure III.8 montre ces différents élements:

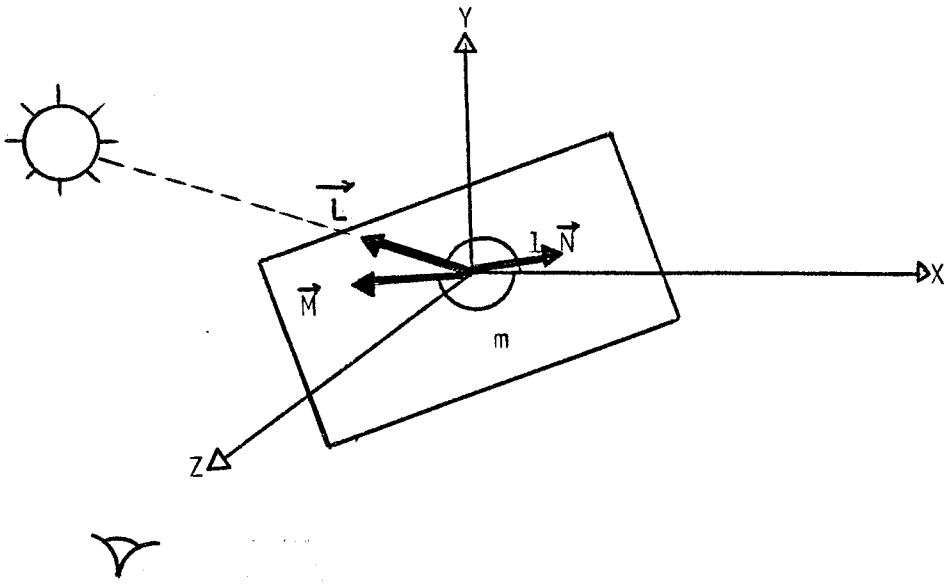


Figure III.8: Géométrie de la réflexion

Ce calcul a nécessité la réalisation de deux opérations élementaires:

- calcul des angles entre deux directions
- modèlisation de la réflexion

* Calcul des angles: l'originalité réside dans la représentation des directions en coordonnées sphériques, comme le schématise la figure III.9 en ce qui concerne la normale à la face

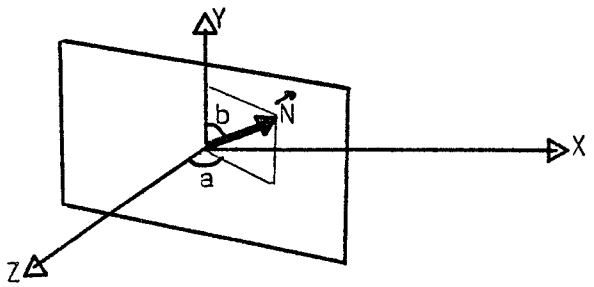


Figure III.9: Représentation en coordonnées sphériques

Chacun des angles a et b est représenté sur 5 bits, et exprimé en 1/32e de pi.

* Modèlisation de la réflexion: quatre modèles sont possibles, mémorisés dans la table des faces, et permettent ainsi différents niveaux de brillance. Chaque modèle est une table donnant le coefficient en fonction de l'angle.

2.4.4. Processeur d'éclairage et d'affichage

Il réalise la synthèse finale en chaque point à partir des données suivantes:

- couleur du point issu des textures
- coefficients de réflexion
- couleur de la source lumineuse et intensité de la lumière ambiante, mémorisées dans les registres du matériel

Le modèle d'éclairage utilisé est le suivant:

$$C = ((A + Rd) * T + Rs * B) * S$$

C : couleur finale
A : intensité lumière ambiante
T : couleur issue de la texture
B : blanc pur
S : couleur source lumineuse
Rd,Rs:coefficients de réflexion

En conclusion à ce paragraphe, nous donnerons en figure III.10 un schéma général donnant l'architecture et la circulation des informations au sein d'HELIOS (MAF82).

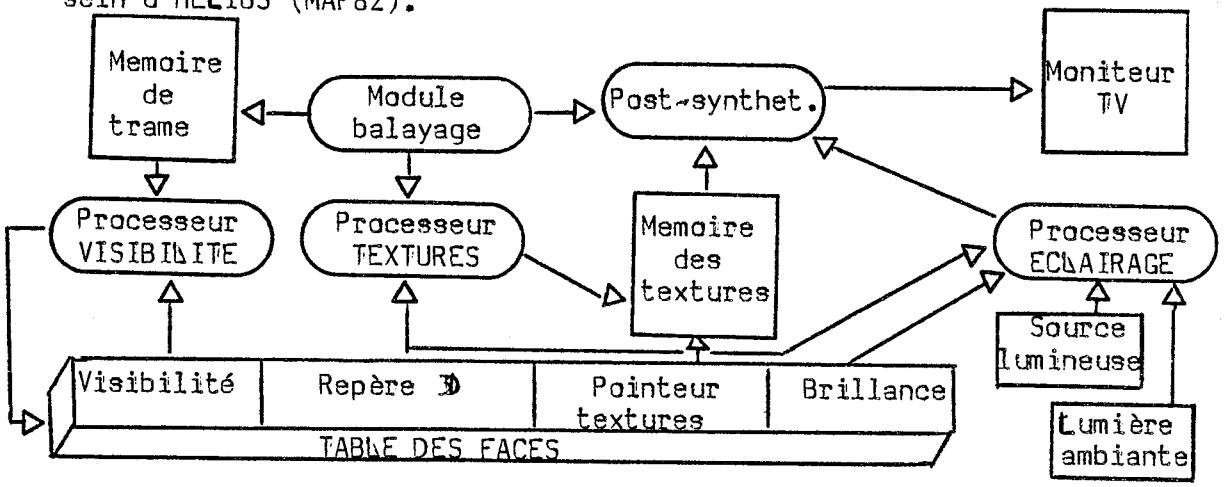


Figure III.10: Architecture des processeurs

2.5. Les différentes versions d'HELIOS

Le prototype HELIOS n'est pas resté figé dans la configuration décrite en paragraphe 2.1. de ce chapitre mais a au contraire évolué, dans un premier temps vers l'utilisation de microprocesseurs, dans un second temps vers une architecture générale quelque peu différente.

2.5.1. Utilisation de microprocesseurs

* CPU MOTOROLA 6809: cette version réalisée en 1982 et utilisée par le laboratoire d'Intelligence Artificielle de l'IMAG, a permis une amélioration des performances mais n'a pas élargi le champ des possibilités d'HELIOS -au contraire pour les besoins de l'application, la notion de texture n'a pas été utilisée- . Le CPU 6809 assure en particulier la gestion des commandes, la communication avec le calculateur et aussi des opérateurs tels que éclairage, gestion du réticule ... assurés par la logique cablée dans la version précédente -version 1 d'HELIOS-. Cette version -version 2- a donc effectué un équilibrage nouveau des tâches entre les différents niveaux que sont le calculateur pilote MICROENGINE, le CPU 6809 et la logique cablée -cf figure III.11-.

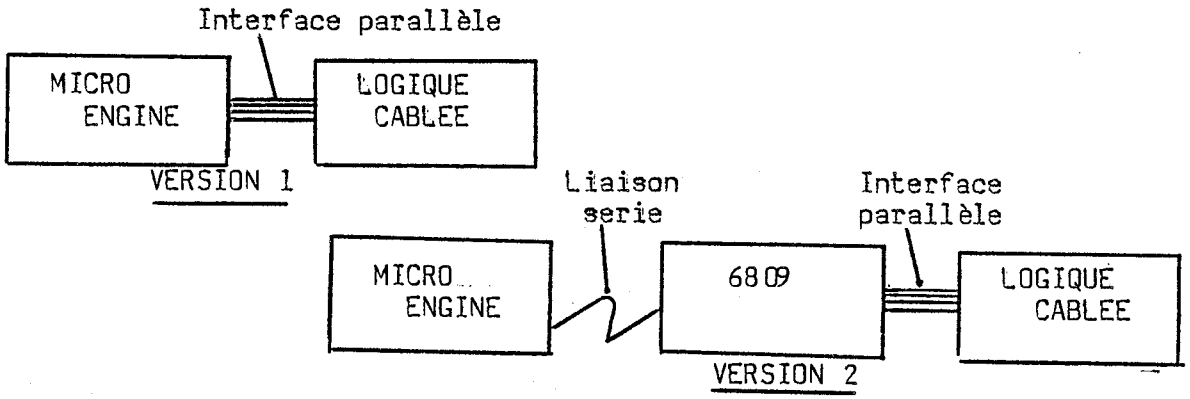


Figure III.11: Parallèle entre versions 1 et 2 d'HELIOS

* CPU 68000: cette version 3 réalisée en 1983-84 reprend les objectifs de la version précédente -amélioration des performances grâce à un haut degré de parallèlisme- et y ajoute une extension à de nouvelles tâches grâce à ses capacités de mémorisation -en particulier traitement des images 3D avec conservation de la structure-. Le problème principal de cette version réside là encore dans le partage des tâches entre les différentes couches, puisque l'on peut imaginer par exemple qu'entre l'interface que constitue le MC68000 avec le calculateur principal et la logique cablée, peut s'insérer une couche de "nanologiciel" à base de microprocesseurs en tranches. La figure III.12 donne la structure possible d'HELIOS avec le MC68000, ainsi qu'un exemple de découpage de tâches pour le traitement d'un polygone.

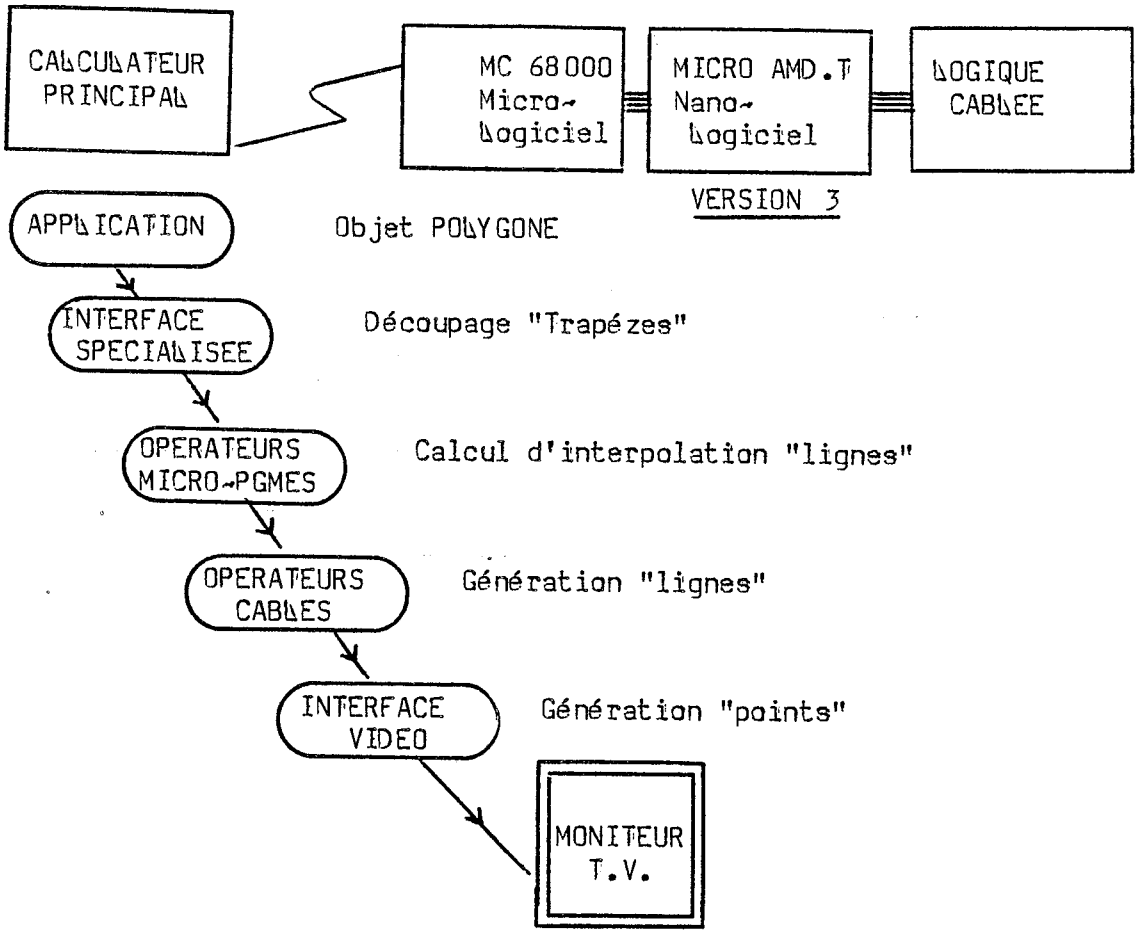


Figure III.12: Exemple d'utilisation de la version 3 d'HELIOS

Ces problèmes font l'objet d'une étude réalisée par K.CHIBANE dans le cadre d'une thèse de 3e. cycle (CHI83).

2.5.2. Architecture banalisée

Cette philosophie d'architecture qui n'affecte que la logique cablée d' HELIOS permet à partir d'unités de traitement banalisées connectées en fond de panier sur un bus standardisé de répondre par des combinaisons à une grande diversité d'applications. Les différentes configurations possibles selon les processus de visualisation envisagés sont pilotées par logiciel. Les problèmes soulevés par ce type d'architecture forment le cadre d'une thèse de 3e. cycle de V.ZARATE (ZAR84), et portent essentiellement sur trois points:

- agencement des bus et modules

- banalisation des modules

- synchronisation des opérations

Des éléments précis concernant cette version 4 d'HELIOS ont été apportés par F.MARTINEZ lors du premier colloque Image de Biarritz en Mai 1984 (MAR84) -cf ANNEXE 1-.

2.5.3. Autres approches

D'autres approches sont envisagées avec en particulier la notion de "station de travail" -workstation-, notion sur laquelle nous reviendrons en toute fin de ce mémoire. Toute cette évolution prouve, s'il en est besoin, les progrès et les améliorations possibles pour cette couche de synthétiseurs, manifestement primordiale dans le domaine de la synthèse d'images.

3. Les synthétiseurs programmables : CLOVIS

Ce paragraphe présente les différents composants logiciels du système de synthèse, tel qu'il a été présenté en début de ce chapitre.

La réalisation du synthétiseur cablé HELIOS a permis d'élargir la gamme des éléments concernés -prise en compte d'attributs d'aspect, d'éclairage, structuration des objets, ... entrainant donc un bon réalisme- et remet en cause ainsi les logiciels graphiques "classiques", cette dénomination recouvrant l'unité U(S,M) dans une organisation en couches -cf. paragraphe 2.3.2. du chapitre II-.

Ces problèmes ont été étudiés dans le cadre du projet CLOVIS, dont l'objectif est la réalisation d'un Complexe Logiciel de Visualisation Interactive Structurée, et constituant l'environnement de ce mémoire.

Nous évoquerons donc ci-après le logiciel pilote, et avant le logiciel principal, les logiciels de base graphiques existants.

3.1. Le logiciel pilote

Nous ne donnerons ici que quelques éléments très succints, en rappelant que la conception et la réalisation de ce logiciel, dans sa version initiale destinée au premier prototype HELIOS, ont fait l'objet d'un mémoire d'Ingénieur C.N.A.M. présenté par M.T.SARRASIN (SAR82). D'autre part, nous reviendrons plus en détail sur ce logiciel dans le chapitre V de ce mémoire.

Nous indiquerons donc simplement l'organisation de ce logiciel, de type hiérarchique, et les différentes fonctions de chaque unité. La figure III.13 schématise l'organisation hiérarchique du logiciel pilote.

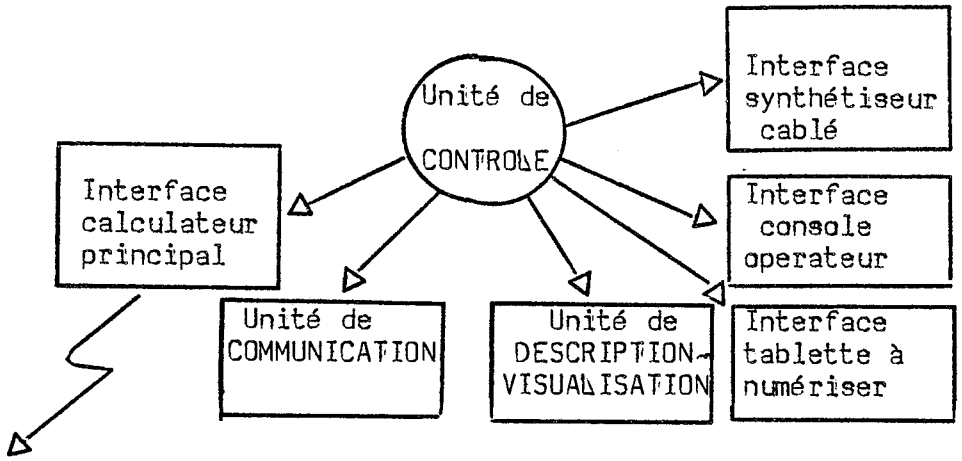


Figure III.13: Organisation hiérarchique du logiciel pilote

* L'unité de contrôle permet trois modes de fonctionnement:

- "connexion": le logiciel se résume à l'échange entre le calculateur principal et la console opérateur

- "opération": fonctionnement normal où tout est commandé par le calculateur principal, sous la responsabilité du logiciel pilote

- "local": fonctionnement interactif à partir de la console opérateur, le calculateur principal n'intervenant plus

* L'unité de communication assure la gestion de la structure de données locale plus simple que celle du logiciel principal que nous verrons ultérieurement dans ce chapitre. Chaque élément est identifié par un entier exprimé sur quatre octets. Le support d'information est une unité de disques souples, comprenant outre le fichier graphique, les banques de textures et de modèles de réflexion.

* Les unités d'interface avec le calculateur principal et le synthétiseur cablé gèrent les échanges pour les quatre types de processus: attribution, consultation, description, visualisation.

* Les interfaces avec les dispositifs de dialogue permettent de gérer:

- le réticule intégré au synthétiseur cablé

- le réticule intégré à la console opérateur

- la tablette à numériser

* Enfin le logiciel pilote permet des fonctions locales d' initialisation -plans d'identification, attributs des faces, source lumineuse, banques des textures et de réflexion-, de configuration en fonction de l'application -remplissage des textures et modèles de réflexion appropriés-, et d'archivage local d'images -mémorisation et récupération-.

3.2. Les principaux logiciels de base graphiques existants

Avant de présenter le logiciel principal de CLOVIS, et afin d'en montrer les spécificités, il est bon de développer quelque peu les réalisations antérieures et les tendances actuelles des logiciels de base. Pour ce, nous nous appuierons sur les travaux réalisés par M.LUCAS lors de sa thèse d'Etat (LUC77), et évoquerons successivement la structure de ces logiciels , l'étude de quelques-uns d'entre eux, et les remarques qui peuvent être apportées.

3.2.1. Structure des logiciels de base

Les logiciels de base réalisés et que nous nous proposons d'étudier répondent aux besoins de systèmes généraux. Ces systèmes se présentent en plusieurs couches, chacune d'elles comportant une bibliothèque d' algorithmes, une stucture de données associée et un moniteur d' enchainement (LUC77). Le logiciel graphique interactif de base rassemble les deux dernières couches, comme nous l'avions évoqué en introduction du paragraphe 3 de ce même chapitre. La figure III.14 représente une telle organisation.

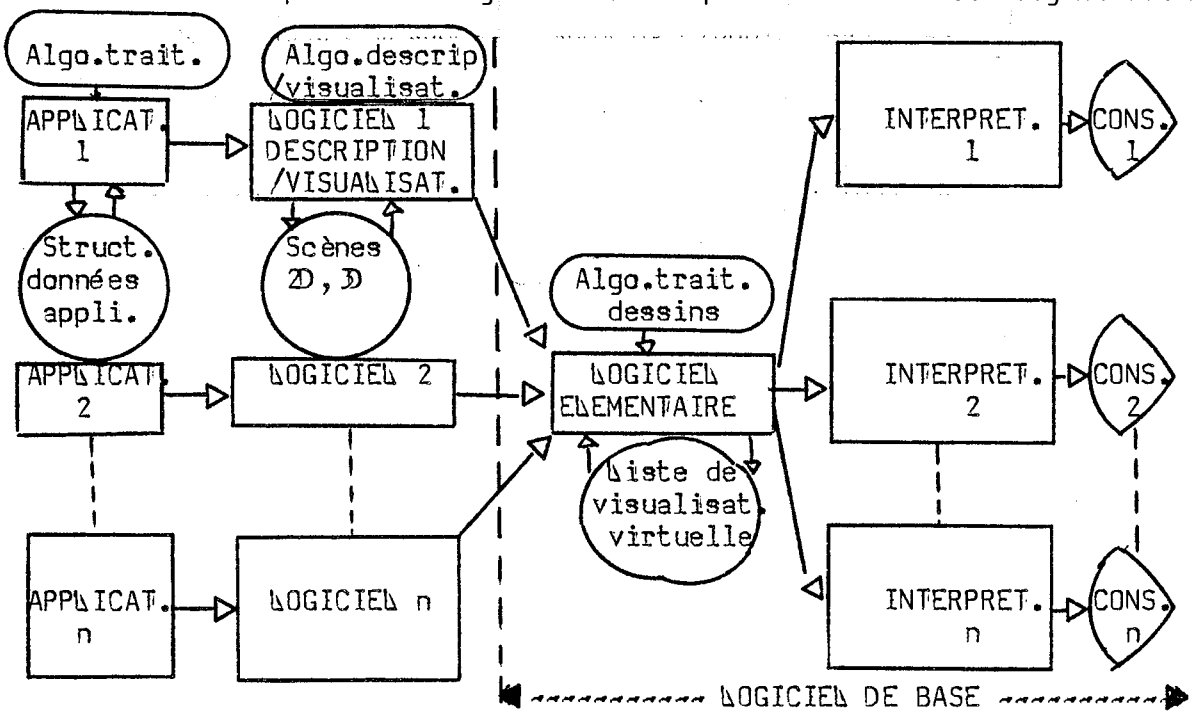


Figure III.14: Organisation en couches

On remarque que dans ce type d'organisation, le logiciel élémentaire constitue le noyau commun indispensable et assure l'indépendance avec le matériel par le concept de "console virtuelle" avant que chaque interpréteur ne le convertisse en "console réelle". Il est clair que cette couche apporte une contrainte certaine au système puisque devant satisfaire au maximum de "couples" applications-matériels.

3.2.2. Présentation de quelques logiciels

Parmi les nombreuses réalisations, nous pouvons citer:

- GRAF: "GRaphic Additions to Fortran" (HCY67), extension graphique de Fortran
- Ensemble de NEWMAN et SPROULL (NES74)
- travail du GSPC: "Graphic Standards Planning Commitee" (GSP77), regroupant des propositions de normalisation des logiciels graphiques -norme ACM-
- IGL: "Interactive Graphics Library" (HVI79), bibliothèque de primitives fournie par le constructeur TEKTRONIX
- GKS: "Graphic Kernel System" (EEK79), résultat du travail de normalisation du groupe allemand DIN
- GRIGRI: (LLM78) recherchant une indépendance vis-à-vis du matériel

Ces différents logiciels présentent des faiblesses dans différents domaines que voici:

* dépendance vis-à-vis du matériel: critique pour GRAF et même IGL dont la compétence s'étend certes sur la gamme TEKTRONIX, mais avec la nécessité parfois de ré-écriture de programmes
* défaut de structuration: pour IGL. En ce qui concerne GKS, ou GRAF la structuration se fait par des déclarations de début (OPEN) et fin (CLOSE) et tient compte non des données mais du déroulement du programme. GSPC et GRIGRI ont des niveaux de structuration indépendants du programme mais peu nombreux (Figure et section)
* insuffisance des élements de base: très sensible pour GRIGRI et GRAF (prise en compte uniquement de points, caractères)
* insuffisance des primitives d'attribution: très réduites dans GRAF, pas de transformations géométriques pour GRIGRI, pas de couleur pour GKS
* non configuration possible des processus de description et visualisation pour GRAF, GRIGRI, GSPC, IGL.
* non cohérence et non symétrie entre les processus et les éléments: GKS est en cause eu égard à ses grandes possibilités de configuration.

Ces différents points d'insatisfaction sont justement, on le verra lors de la présentation du logiciel principal de CLOVIS dans le paragraphe suivant, les idées fortes et constituent un souci permanent dans la conception et la réalisation de CLOVIS.

3.3. Le logiciel principal de CLOVIS

Ce logiciel est implanté sur le calculateur principal, et propose directement aux utilisateurs un jeu de primitives via l'unité de contrôle. Sa philosophie est une extensibilité maximale, au fur et à mesure des besoins , autour d'un noyau minimal structuré et consacré aux primitives ayant un rapport direct avec les quatre processus de base.

Nous présentons ci-après les unités de communication et de contrôle de ce logiciel.

3.3.1. L'unité de communication

Elle a pour but essentiel la gestion de la structure de données graphiques , à laquelle nous nous intéresserons tout particulièrement dans ce paragraphe. Cette gestion inclut des actions de structuration, parcours, affectation, et recherche, protège les attributs engrangés dans la structure de données.

* La structure de données arborescente: ce type de structure est guidé par les niveaux de structuration que l'on a pu évoquer jusque là, à savoir un niveau "maquette", un niveau "éléments", et une structuration des éléments eux-mêmes, structuration pouvant traduire des relations géométriques -proximité, inclusion, assemblage- ou fonctionnelles -unité de mouvement, aspect, ...-. Ainsi le fichier graphique a-t-il une structure d'arbre représenté par la figure III.15.

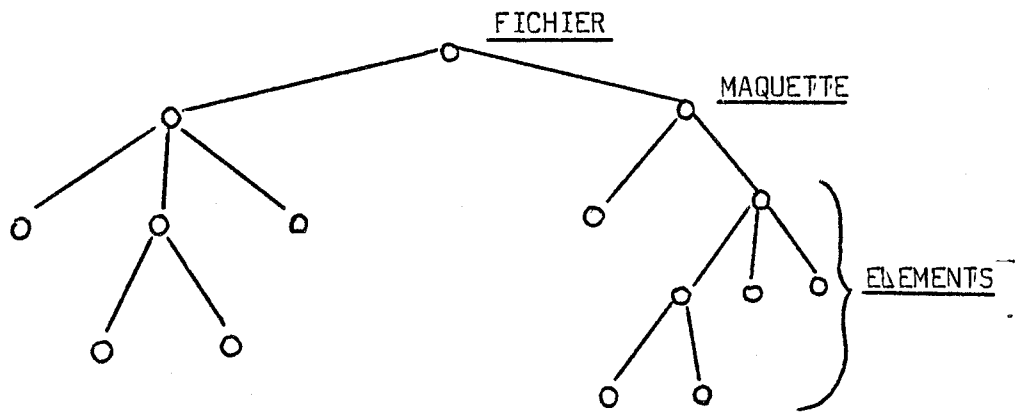


Figure III.15: Structuration du fichier graphique

Les avantages d'une telle structuration répondent bien aux besoins d'un système de synthèse:

- nombre de niveaux illimité, permettant ainsi la manipulation de maquettes complexes

- application d'opérations à un ensemble de noeuds, ce qui assure cohérence et puissance

- compositions possibles sur un sous-arbre issu d'un noeud

- puissance d'interaction, puisqu'à partir de la désignation d'un élément , la sous-structure associée est également concernée

- protection des données assurée par la décomposition possible en sous-arbres

- optimisation de processus, en particulier de visualisation: en effet nombre de techniques utilisent les propriétés des arbres. Citons des algorithmes d'élimination de parties cachées par une structure hiérarchique (CLA76), les techniques du "BSP-tree" (FKN80) ou encore le "Ray casting" (BWJ84) -cf. ANNEXE 2-.

* Les mécanismes de dénomination:

- un codage alphanumérique par un identificateur de 8 caractères au plus a été retenu

- des règles syntaxiques assez élaborées ont été mises en place, permettant pour identifier un noeud,l'utilisation de noms:
  - relatifs: expression du chemin à suivre à partir d'un noeud quelconque
  - absolus: expression de chemin à suivre à partir de la racine du fichier graphique
  - globaux: expression simultanée de plusieurs chemins
  - indicés: expression de sous-arbres identiques différenciés par un indice

Les règles syntaxiques valides peuvent être classées en deux groupes, selon qu'il s'agit d'opérations de construction ou de parcours.

. Règles de construction

- -> Syntaxe:

< structure>::=<ss-structure>(","<ss-structure>)*
< ss-structure>::=<identif>(" "<indices>" ")(" "<structure>" ")

< indices>:=<borne>(":"<borne>)
< borne>::=<-99..99> ou "!"

- -> Exemple -cf. figure III.16-: expression de la construction d'un nombre indéterminé de "vélo", comportant un "cadre", avec une "selle" et un "guidon", et deux "roue" comportant une "jante" et un ensemble de "rayon".

velo[1:!](cadre(selle,guidon), roue[1:2](jante,rayons))

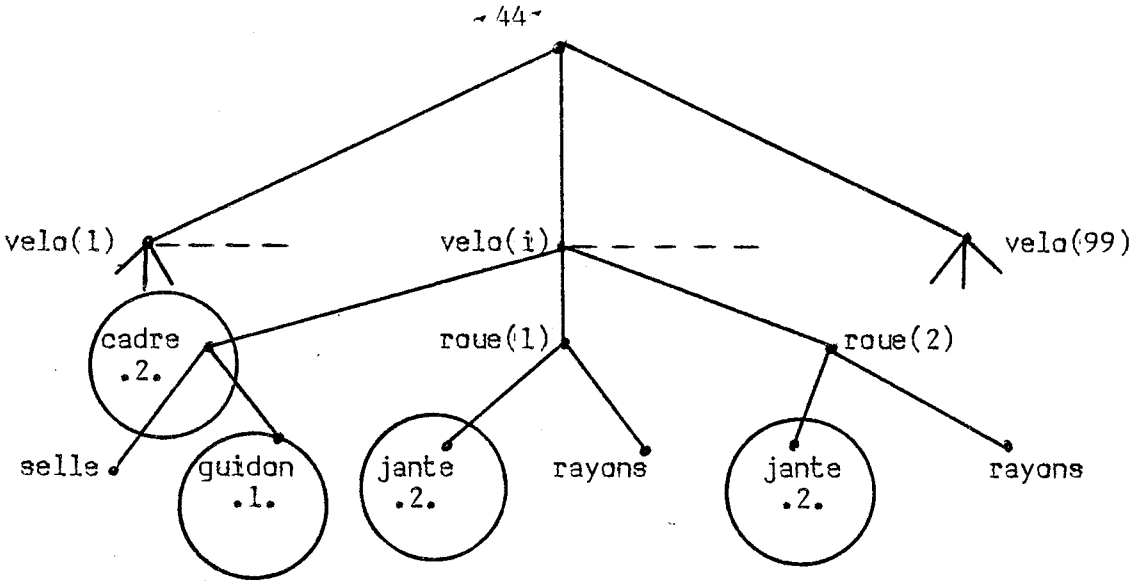


Figure III.16: Exemple de syntaxe

. Règles de parcours: référence à un noeud unique

- -> Syntaxe:

< chaine-simple>::=(<branche>)
< branche>::=<identif>(" "<indice>" ")("("<branche>")")
< indice>::=<-99..99>

- -> Exemple -cf. figure III.16- :l'expression
velo[17](cadre(guidon))
référence le noeud no.1 de la figure, si i est égal à la 17eme occurence de "velo".

. Règles de parcours: référence à plusieurs noeuds

- -> Syntaxe:

< chaine-multiple>::="*" ou <branche>(","<branche>)*
< branche>::=<identif>(" "<indice>" ")("("<chaine-multiple>")")
< indice>::=<borne>(":"<borne>)
< borne>::=<-99..99> ou "*"

- -> Exemple -cf. figure III.16- :
l'expression

velo[1:17](roue[*](jante),cadre)
référence les noeuds no.2 de la figure, pour toutes les occurences de "velo" comprises entre 1 et 17.

* Les primitives de l'unité de communication: comme nous l'avons indiqué dans la présentation de celle-ci, -cf.paragraphe 3.3.1. de ce chapitre- deux groupes de primitives apparaissent:

- structuration et parcours

- affectation et recherche

- Primitives de structuration: elles sont utilisées par l'unité de contrôle mais proposées aussi au programme d'application
  - Construction d'une structure: la fonction

    STRUCTURE (<structure>) : <référence>

    donne en retour l'adresse de la racine de la structure créée.
  - Identification d'une structure : les fonctions

    IDENTITE (<chaine-simple>) : <réf-simple>
    IDENTITE (<chaine-multiple>) : <réf-multiple>

    donnent en retour l'adresse de la racine de la structure décrite et consistent en une analyse syntaxique de la chaine fournie. La primitive

    FINIDENTITE

    permet de revenir à l'identité précédente.
  - Contexte : la primitive

    CONTEXTE (<réf-simple>)

    permet de définir un contexte implicite pour toutes les actions à venir, permettant en particulier de simplifier la syntaxe des chaines d'identification. Les contextes sont empilés, et peuvent être dépilés par la primitive

    FINCONTEXTE

  - Parcours d'une structure: la primitive de parcours, utilisée uniquement par l'unité de contrôle, parcourt la structure à partir de la racine du contexte et invoque pour chaque noeud l'action correspondante.

    PARCOURS (<réf-multiple>, action-A, action-D, action-F)

    réf-multiple:adresse de la structure définie par "IDENTITE"
    action-A:action à appliquer aux noeuds parcourus dans l'ordre ascendant
    action-D:action à appliquer aux noeuds parcourus dans l'ordre descendant
    action-F:action à appliquer aux feuilles de l'arbre

  - Exemple d'utilisation de ces primitives:

    A <-- STRUCTURE (velo[1:10](roue[1:2](jante, rayon)))
    cf. figure III.17

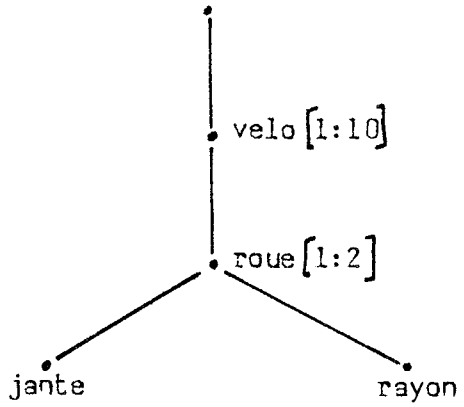


Figure III.17: Exemple de création de structure

B <-- IDENTITE (velo [3] (roue [1] (jante), roue [2] (rayon)))
cf. figure III.18 -trait épais:référence les noeuds no.1 et 2-

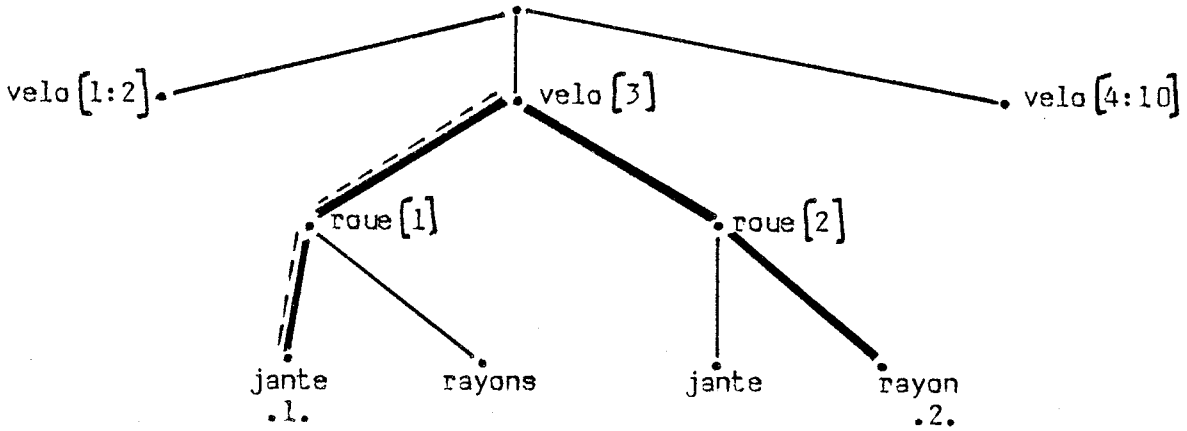


Figure III.18: Exemple d'éclatement d'arbre

CONTEXTE (velo 3 (roue 1))
C <-- IDENTITE (jante)
cf. figure III.18 -trait pointillé:référence le noeud no.1-
FINIDENTITE
FINCONTEXTE
retour au contexte de la figure III.18 -trait épais-
PARCOURS (B, action-F, action-D, action-A)
engendre la séquence d'actions suivantes:
action-D (velo [3])
action-D (roue [1])
action-F (jante)
action-A (roue [1])
action-D (roue [2])
action-F (rayon)
action-A (roue [2])

action-A (velo [3])

- Primitives d'affectation: elles permettent de gèrer les attributs associés aux différents noeuds de la structure. Ce paragraphe présente succintement les éléments retenus pour la modèlisation interne des attributs puis la liste des primitives offertes.

. Modèlisation interne des informations: elle s'appuie sur les quelques remarques suivantes:

- ->pas de modèlisation à priori puisque la structure de données doit être banalisée

- ->le nombre d'attributs pour chaque noeud est inconnu -fixé à 7 maximum-

- ->certains attributs peuvent être attachés simultanément à plusieurs noeuds

En découle une organisation décrite par la figure III.19.

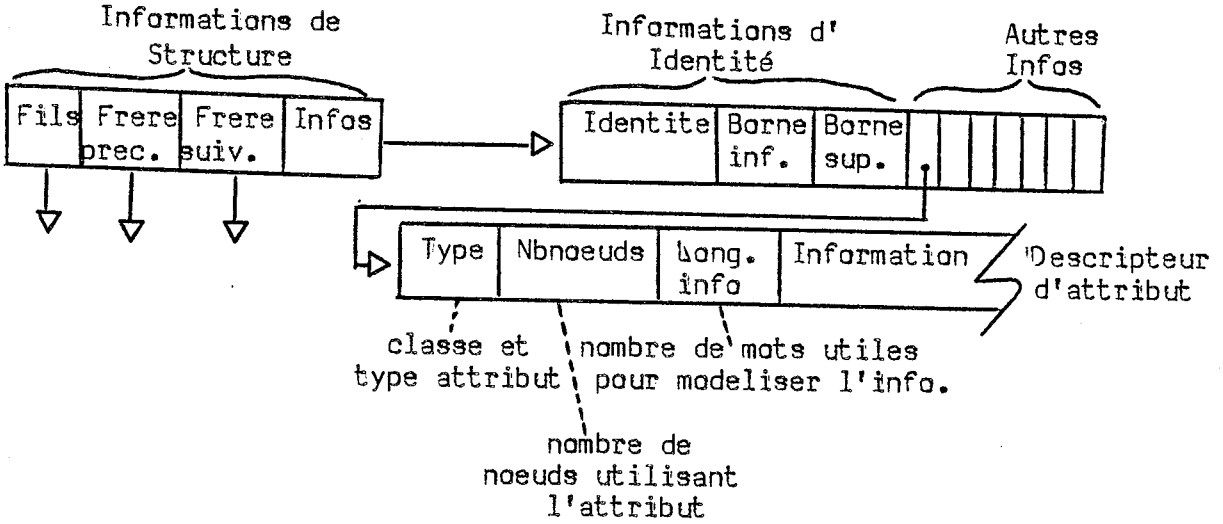


Figure III.19: Modèlisation des attributs pour chaque noeud

Il est à noter d'autre part que si certains attributs peuvent rester inchangés, d'autres au contraire sont variables. Aussi afin de minimiser les échanges entre application et système, nous distinguons pour chaque élément les attributs constants conservés par le système et déchargeant l'application et les attributs variables conservés par l'application et examinés le cas échéant par le système. -cf.figure III.20-.

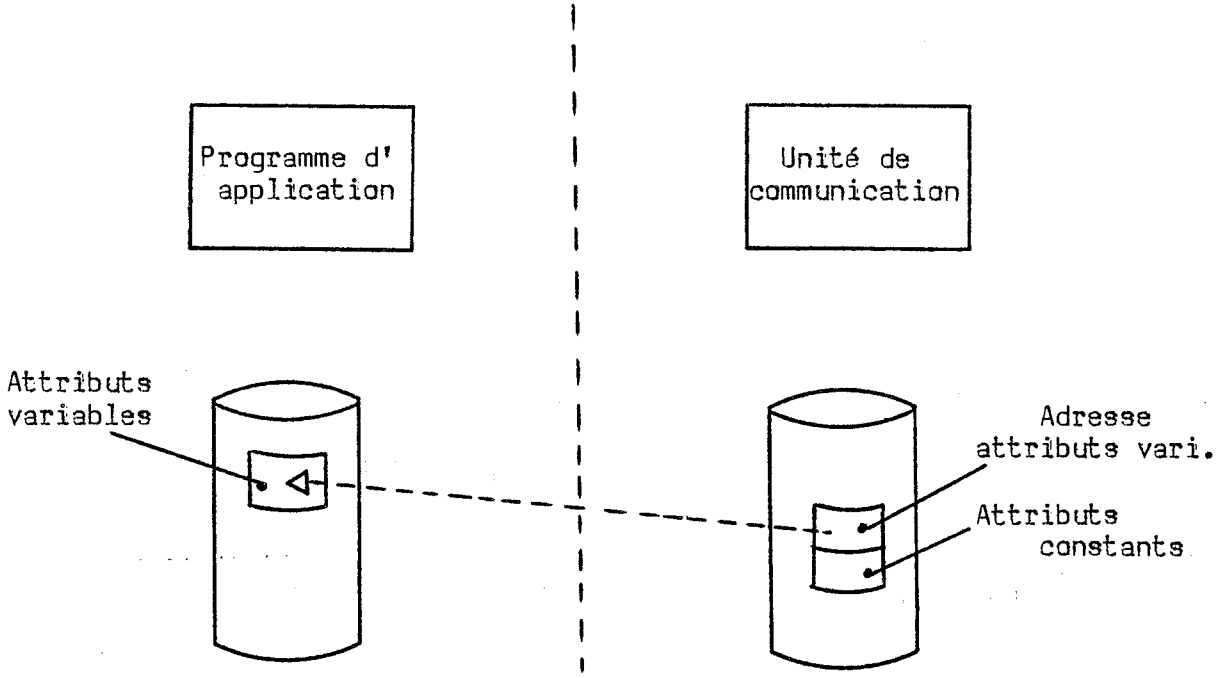


Figure III.20: Attributs variables et constants

. Les primitives offertes d'affectation et de recherche sont utilisées uniquement par l'unité de contrôle

- -> Affectation d'un attribut

AFFECTER (ref-simple, type-attribut, adresse-info, cte/var)

Les types d'attributs sont des fonctions assez évoluées afin de décharger l'application de modèlisations de certains types d'éléments.

- -> Recherche d'un attribut

INFO (ref-simple, type-attribut) : adresse-info

- -> Destruction d'un attribut

DETRUIRE (ref-simple, type-attribut)

- -> Enfin, afin d'optimiser les processus, deux fonctions logiques permettent de tester l'état de chaque attribut d'un noeud donné:

EXISTE : vrai si l'attribut existe
MODIF : vrai si l'attribut a été modifié

3.3.2. L'unité de contrôle

Cette unité a pour but, rappelons-le, de gèrer et d'élaborer les quatre processus fondamentaux. Pour ce, elle dispose des opérateurs suivants:

- opérateurs de l'unité de communication:
  STRUCTURE, IDENTITE, CONTEXTE, PARCOURS
  AFFECTER, INFO , DETRUIRE
  EXISTE, MODIF

- opérateurs de l'unité de description et visualisation que nous étudierons au cours des chapitres suivants

- opérateurs du logiciel pilote
  ATTRIBUER-pilote
  CONSULTER-pilote
  VISUALISER-pilote
  DECRIRE-pilote

Elle propose donc essentiellement à l'utilisateur les primitives suivantes:

STRUCTURE, IDENTITE, CONTEXTE
ATTRIBUER
CONSULTER
VISUALISER
DECRIRE

Ces quatre dernières étant l'objet des lignes suivantes.

* La'attribution

ATTRIBUER (ref-multiple, type-attribut, adr-info, cte/var)

Cette primitive parcourt la structure dont la racine est "ref-multiple", et provoque l'affectation aux noeuds de la structure et leur marquage.

* La consultation

CONSULTER (ref-simple, type-attribut)

* La visualisation

VISUALISER (ref-multiple, processus)

Elle parcourt l'arbre référencé par "ref-multiple", détermine les attributs modifiés, ou détruits depuis la dernière visualisation, recalcule les éléments modifiés et communique au calculateur pilote les éléments de traitement.

* La description

Elle peut être explicite, c'est-à-dire invoquant un processus de description à l'aide de la primitive:

DECRIRE (ref-multiple, type-attribut, processus)

Ce processus peut faire appel à une construction à l'aide de dispositifs de collecte -tablette à numériser par exemple- ou à une référence à un attribut d'un élement désigné directement sur écran.

Mais la description peut aussi être implicite, lorsqu'elle est effectuée systématiquement si un attribut est incomplet.

3.4. Conclusion

C'est autour de ce logiciel principal que s'articulent les deux derniers chapitres de ce mémoire. Ils présentent de facon plus détaillée le logiciel réalisé à partir des éléments exposés jusque là.

Le chapitre IV précisera le cadre dans lequel le logiciel a été réalisé, les techniques de gestion de la structure de données impliquée dans l'unité de communication, et la définition des éléments et primitives manipulés au sein de l'unité de contrôle.

Quant au chapitre V, il s'attache directement à la réalisation du logiciel, indiquant le contexte purement matériel, la structure du logiciel et son mode d'utilisation, et les différentes fonctions réalisées tant dans l'unité de contrôle que celles de description-visualisation et de communication.

CHAPITRE IV : PRESENTATION DU LOGICIEL A REALISER ASPECTS UTILISATEUR ET SYSTEME

Ce chapitre, présentant le logiciel de base implanté sur le calculateur principal et conforme aux notions de base énoncées dans le chapitre précédent, s'articule autour de trois axes :

- le cadre précis de réalisation, tant au plan de la configuration matérielle que des objectifs à atteindre
- l'aspect utilisateur, définissant d'une part les éléments manipulés, d'autre part les primitives ou commandes disponibles et offertes à l'utilisateur
- l'aspect système, où sera précisée la gestion de la structure de données arborescente, tâche essentielle dédiée à l'unité de communication, et où, dans un second temps, seront examinés les facteurs influant la gestion des processus, tâche concernant l'unité de controle

Ainsi sera définie l'ossature du logiciel, et nous terminerons par l'énoncé des problèmes soulevés par cette réalisation.

1. Cadre de réalisation du logiciel

1.1. Configuration matérielle

La réalisation de ce logiciel fait partie prenante du complexe logiciel principal CLOVIS. Il s'agit plus précisément du logiciel principal -cf paragraphe 3.3 du chapitre III- implanté sur le calculateur hôte du système, qui est en l'occurence un VAX 11/780 situé dans les murs de la société MICADO, implantée dans la ZIRST de MEYLAN. La philosophie de ce logiciel étant, rappelons-le, une adéquation optimale entre le plus grand nombre d'applications et le maximum de matériels, ce logiciel a été réalisé en deux versions, l'une destinée à un TEKTRONIX 4114 à vocation 'Dessin au trait' -travail réalisé par J.F. GRABOWIECKI dont plusieurs documents internes à l'équipe (GRA84) en ont rendu compte-, l'autre destinée au terminal de synthèse d'images HELIOS et présentée ici. Comme précisé dans le chapitre précédent -cf paragraphe 2.5 du chapitre III-, HELIOS comporte différentes architectures et ici s'imposent plusieurs précisions.

Le logiciel a été réalisé dans l'optique de l'utilisation d'un terminal muni d'un microprocesseur -tel que 6809 ou 68000-, assurant donc, outre une partie du logiciel pilote, la gestion des échanges avec le calculateur principal, assimilant ainsi HELIOS à un terminal "classique" géré en Xon/Xoff. Mais une telle configuration d'HELIOS n'étant pas disponible au laboratoire, c'est sur le prototype 1 qu'a été effectivement réalisé le logiciel -version munie du micro-calculateur MICRO-ENGINE de

WESTERN DIGITAL-, nécessitant l'écriture d'une interface sur ce matériel simulant la version 2 -interprétation des commandes et communication-, sans bien sûr l'obtention des mêmes performances. Ce fut donc le premier point du travail, amenant à une configuration de tests schématisée par la figure IV.1

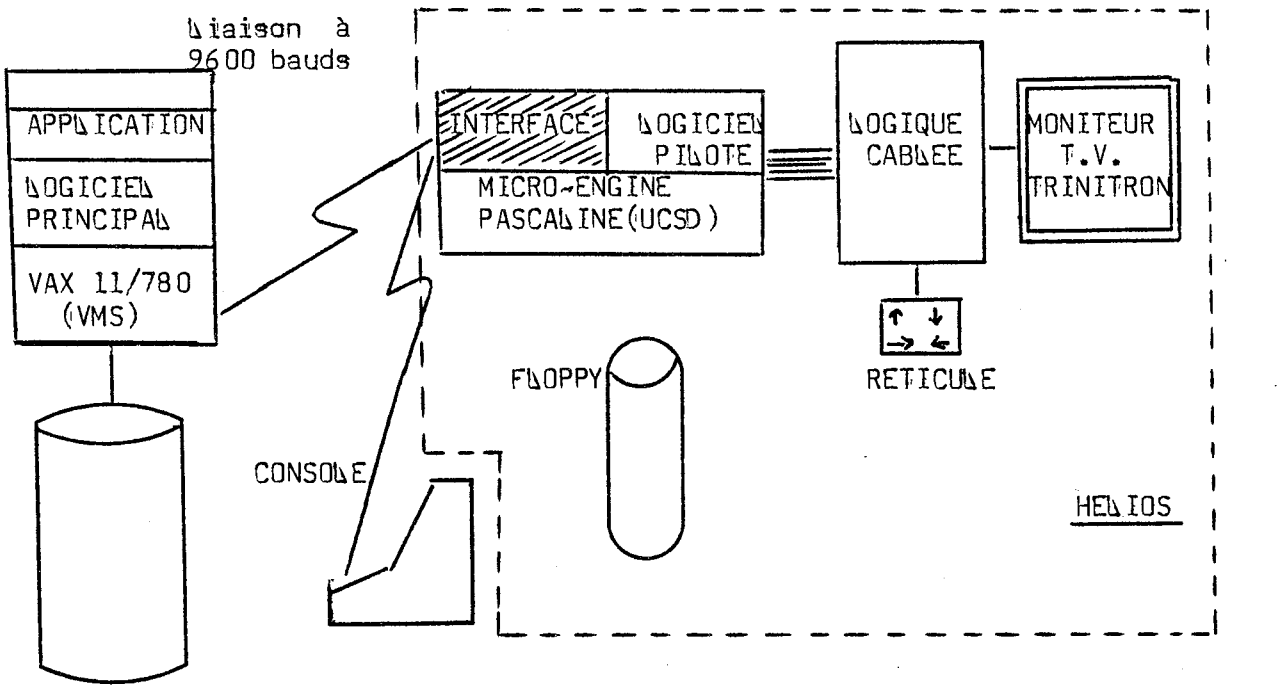


Figure IV.1: Configuration de réalisation

1.2. Objectifs du logiciel

Le logiciel principal, implanté sur VAX et utilisant une structure de données graphiques arborescente conforme au projet CHOVIS, permet le pilotage du terminel HELIOS dont les mots-clés sont images réalistes et temps réel. Il a comme optique de décharger au maximum l'utilisateur et de compléter et exploiter le plus efficacement possible les possibilités du matériel.

L'élément de base manipulé par ce logiciel est l'objet graphique évolué -2D ou 3D-, le paragraphe 2 de ce chapitre analysant tous les types permis à ce jour par le système. Nous aurons l'occasion de découvrir les problèmes posés par la relation entre cette notion d'objet et l'élément de base traité par le matériel, en particulier la face pour HELIOS -cf paragraphe 2.3 du chapitre III-.

La conception et la réalisation du logiciel doivent répondre à plusieurs soucis :

- lisibilité et fiabilité, tout spécialement en vue d'une commercialisation

- portabilité et modularité, en vue d'extensions tant matérielles -implantation sur d'autres calculateurs, adaptation à d'autres terminaux graphiques- que logicielles -ajouts de primitives et d'éléments, prise en compte de nouvelles applications-.

- structuration logique, respectant l'organisation hiérarchique du logiciel et facilitant là aussi l'insertion de nouveaux processus par le concepteur.

Enfin, la difficulté première du logiciel réside sans aucun doute dans l'équilibrage des tâches allouées aux différentes couches -logiciel principal,logiciel pilote, logique cablée-.

Ces éléments seront approfondis lors de l'étude des problèmes rencontrés en cours de réalisation en fin de ce chapitre.

2. Point de vue de l'utilisateur

2.1. Liste et classification des types d'attributs

Comme nous l'avons vu dans le chapitre I présentant la synthèse d'images, les informations peuvent être regroupés en 6 classes :

- Identité
- Morphologie
- Aspect
- Géométrie
- Eclairage
- Structure

Nous nous attacherons ici à présenter les composantes des classes M, A, G, E, les deux classes restantes étant décrites essentiellement par la structuration des données.

2.1.1. Morphologie

Les attributs morphologiques, déterminant donc la forme intrinsèque de chaque objet, se décomposent en éléments pleins -surface- et éléments fil de fer, en éléments 2D et éléments 3D, en éléments plans et éléments gauches. Certains types peuvent être regroupés sur des critères de description et constituent un groupe homogène d'attributs s'adressant à des mêmes primitives.

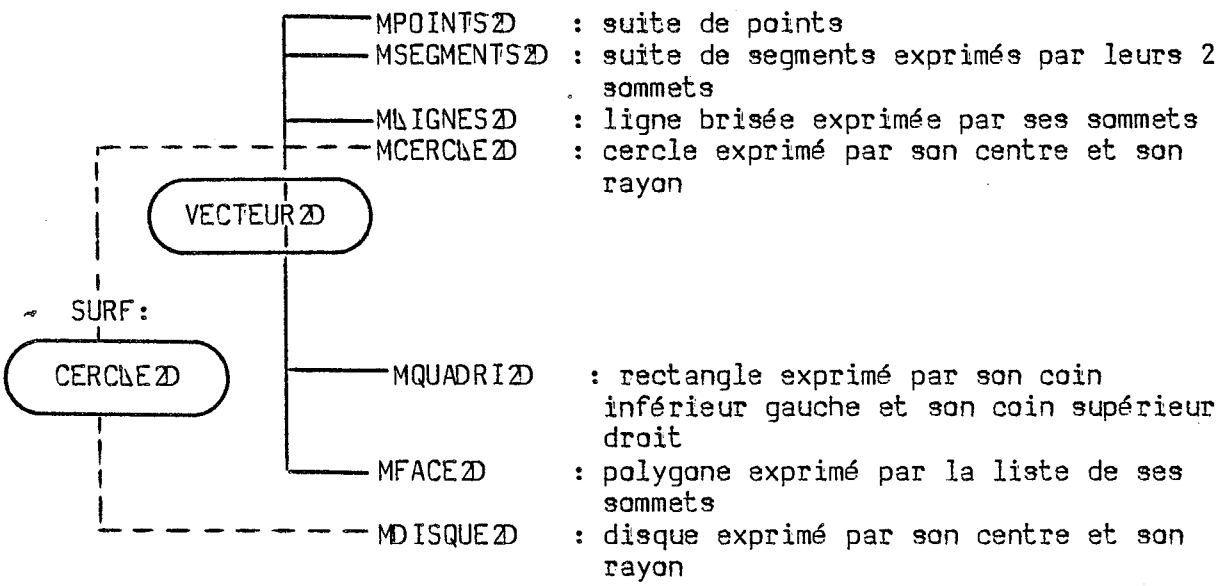
* Eléments "CARACTERES"

- FIL DE FER: ('TEXTEFDF')
  TD: suite de caractères ASCII en D
  T3D: suite de caractères ASCII en 3D

- SURF: ('TEXTESURF')
  TEXTED: suite de caractères D pleins
  TEXTE3D: suite de caractères 3D pleins

* Eléments "POINTS D"

- FIL DE FER:



Dans cette catégorie d'éléments, une classification selon le type de description dégage deux ensembles :
.VECTEURD (MPOINTSD, MSEGMENTSD, MLIGNESD, MQUADRID, MFACED)
.CERCLED (MCERCLED, MDISQUED)

* Eléments "POINTS 3D"

- FIL DE FER :

MPOINTS3D
MSEGMENTS3D
MLIGNES3D
} même description qu'en D

- SURF :

MFACE 3D : surface plane dans l'espace exprimée par le ou les polygones composant son contour
MFONCTION 3D : surface gauche exprimée par une fonction à 2 variables

2.1.2. Aspect

Trois groupes de types d'attributs émergent dans cette classe: ACAR -aspect caractères-, ATRAIT -aspect fil de fer-, ASURF -aspect plein-.

* ACAR :

- taille des caractères: exprimée par la hauteur, la largeur et l'espace entre deux caractères en nombre de points
- police des caractères: standard ou spécialement définie par l'utilisateur
- écriture: normale ou italique
- orientation des caractères: horizontale, verticale, diagonale ascendante, diagonale descendante
- couleur des caractères
- visibilité, et clignotement des caractères

* ATRAIT :

- couleur du trait
- visibilité et clignotement du trait
- texture du trait: continu, pointillés, pointillés rapprochés, tirets courts, tirets longs, trait mixte court, trait mixte long, et pas de trait

* ASURF :

- couleur et texture prédéfinie
- visibilité, clignotement et transparence
- modèle de réflexion: neutre, mat, métallique, brillant, ou prédéfini
- mode de remplissage: uniquement le contour de la face -fermé ou non-, remplissage total avec une texture

2.1.3. Géométrie

Deux types d'attributs géométriques sont permis selon le dimension souhaitée (2D ou 3D)

* G2D : géométrie d'un élément bi-dimensionnel exprimée par une matrice 3x3 en coordonnées homogènes

* G3d : géométrie d'un élément tri-dimensionnel exprimée par une matrice 4x4 en coordonnées homogènes

2.1.4. Eclairage

Cette classe ne comporte qu'un type d'attribut, ne concernant que les types d'éléments morphologiques du groupe "SURF"

* ECL :

- couleur de la source exprimée en (RVB)

- intensité de la lumière ambiante

- direction de la source exprimée en coordonnées sphériques

A ces informations propres à l'élément manipulé, peuvent s'ajouter deux points pouvant influer la modèlisation de l'éclairage et concernant les faces d'HELIOS -cf paragraphes 2.3 et 2.4.3 du chapitre III- :

- vecteur normal à la face, exprimé en coordonnées sphériques

- repère associé à la face en coordonnées cartésiennes

Restent deux classes d'attributs, que nous n'avons pas citées en tête de ce paragraphe, concernant des éléments géométriques de prise de vue et d'affichage.

2.1.5. Géométrie de la prise de vue

De même que pour la géométrie, deux types sont proposés selon la dimension des éléments manipulés :

* GV2D :

- matrice 3x3 de transformations

- limites du domaine visible: coins inférieur gauche et supérieur droit de la fenêtre

* GV3D :

- matrice 4x4 de transformations

- limites du domaine visible

2.1.6. Géométrie de l'affichage

Un seul type compose cette classe

* Ga :

- limites de la cloture : coins inférieur gauche et supérieur droit

2.1.7. Récapitulatif

Cette énumération des types d'attributs nous permet de dresser un tableau récapitulatif -cf figure IV.2- dégageant les corrélations entre les différentes classes et ainsi la liste complète des types d'éléments manipulés par le logiciel, définis comme l'association de chacune des classes, la classe morphologique étant prise comme "leader".

AUTRES CLASSES / CLASSE M	A			G		E	Gv		Ga
	ACAR	ATRAIT	ASURF	G2D	G 3D	ECL	Gv2D	Gv 3D	Ga
T2D	X			X	X		X		X
.F.									
T 3D	X				X			X	X
---.1.---									
TEXTE2D	X		X	X	X	X	X		X
.S.									
TEXTE 3D	X		X		X	X		X	X
MPOINTS2D		X		X	X		X		X
MLIGNES2D		X		X	X		X		X
.F.									
MSEGMENTS2D		X		X	X		X		X
MCERCLE2D		X		X	X		X		X
---.2.---									
MQUADRI2D		X	X	X	X	X	X		X
MFACE2D		X	X	X	X	X	X		X
.S.									
MDISQUE2D		X	X	X	X	X	X		X
MPOINTS 3D		X			X			X	X
MLIGNES 3D		X			X			X	X
.F.									
MSEGMENTS 3D		X			X			X	X
---.3.---									
MFACE 3D		X	X		X	X		X	X
.S.									
MFONCTION 3D		X	X		X	X		X	X

.1. = CARACTERES

.2. = POINTS2D

.3. = POINTS 3D

.F. = FINDEFER

.S. = SURFACE

Figure IV.2: Corrélation entre types d'attributs

2.2. Liste des primitives et commandes

L'utilisateur a à sa diposition un ensemble de primitives classées en grands groupes : -cf paragraphe 3.3.2 du chapitre III-

- Structuration
- Attribution
- Consultation
- Visualisation
- Description

Ces primitives, invoquées dans le programme d'application peuvent également être utilisées en mode conversationnel, sous forme de commandes à partir d'un menu.

Nous indiquerons dans un premier temps la liste des primitives avec leurs paramètres associés, dans un second temps un tableau récapitulatif des commandes, les paramètres étant identiques.

2.2.1. Les primitives de structuration

* STRUCTURES (chainecar) : "accroche" à l'identité courante la structure définie par "chainecar"
* IDENTITE (chainecar) : précise l'identité courante pour les actions à venir
* FINIDENTITE : dépile l'identité courante
* IDENTITEPHEMERE (chainecar) : précise l'identité courante pour l'action à venir. Il s'agit d'une facilité équivalente à IDENTITE + FINIDENTITE
* CONTEXTE (chainecar) : précise le contexte courant pour les actions à venir
* FINCONTEXTE : dépile le contexte courant
* DETRUIRE : détruit la structure dont l'identité courante est la racine
* EFFACER : marque la structure dont l'identité courante est la racine, pour non prise en compte de cette structure lors du processus de visualisation.

2.2.2. Les primitives d'attribution

- Attribution MORPHOLOGIQUE :

  * ATTRIBUTVECTEUR2D (morpho, liste-coord.) :
    attribue à l'identité courante un attribut morphologique de type "morpho" dans la sous-classe VECTEUR2D -cf paragraphe 2.1.1 de ce même chapitre-, et décrit par les coordonnées de ses sommets "liste-coord."
    Les valeurs de "morpho" sont des entiers :
    - MPOINTS2D (74)
    - MSEGMENTS2D (75)
    - MLIGNES2D (76)
    - MQUADRI2D (77)
    - MFACE2D (78)

  * ATTRIBUTCERCLE2D (morpho, p1, p2, p3)
    attribue à l'identité courante un attribut morphologique de type "morpho" dans la sous-classe CERCLE2D -cf paragraphe 2.1.1- décrit par son centre de coordonnées (p1, p2) et son rayon (p3).
    Les valeurs de "morpho" sont des entiers :
    - MCERCLE2D (79)
    - MDISQUE2D (80)

  * ATTRIBUTEXTEFDF (morpho, p1, p2, long-chaine, p3) :
    attribue à l'identité courante un attribut morphologique de type "morpho" dans la sous-classe TEXTEFDF -cf paragraphe 2.1.1. décrit par son point initial de coordonnées (p1, p2), la longueur du texte en nombre de caractères "long-chaine", et le texte p3.
    Les valeurs de "morpho" sont des entiers :
    - MT2D (70)
    - MT3D (71)

- Attribution d'ASPECT :

  * ATTRIBUTASPECT (aspect, code) :
    attribue à l'identité courante un attribut d'aspect de type "aspect" et de valeur "code".
    Les valeurs de "aspect" sont des entiers :
    - ACAR-COULEUR (100)
    - ACAR-VISIBILITE (101)
    - ACAR-TAILLE (102)
    - ACAR-POLICE (103)
    - ACAR-ECRITURE (104)
    - ACAR-ORIENTATION (105)
    - ATRAIT-COULEUR (110)
    - ATRAIT-VISIBILITE (111)
    - ATRAIT-GRAPHIQUE (112)
    - ASURF-COULEUR (120)
    - ASURF-VISIBILITE (121)
    - ASURF-TEXTURE (122)
    - ASURF-REFLEXION (123)
    - ASURF-REMPLISSAGE (124)

Les valeurs de "code" sont des entiers :

- > COULEUR, TEXTURE : valeurs de 1 à 495, déterminant les pointeurs sur la mémoire de textures.

- > VISIBILITE

1 INVISIBLE
2 VISIBLE
3 CLIGNOTANT
4 SEMI-TRANSPARENT

- > TAILLE

1 TAILLE 0
2 NORMALE
3 2
4 3
5 4
6 5
7 6
8 7

- > POLICE

1 STANDARD
2 SPECIAL

- > ECRITURE

1 NORMALE
2 ITALIQUE

- > ORIENTATION

1 HORIZONTALE
2 VERTICALE
3 DIAGONALE ASCENDANTE
4 DIAGONALE DESCENDANTE

- > GRAPHIQUE

1 PAS DE TRAIT
2 TRAIT CONTINU
3 POINTILLES
4 POINTILLES RAPPROCHES
5 TIRETS COURTS
6 TIRETS LONGS
7 MIXTE COURT
8 MIXTE LONG

- > REFLEXION

1 NEUTRE
2 MAT
3 BRILLANT
4 METALLIQUE

- > REMPLISSAGE
  1 CONTOUR
  2 CONTOUR FERME
  3 REMPLI

- Attribution GEOMETRIQUE : (G2D)

  * TRANSLATION (XD, YD, XF, YF)
    attribue à l'identité courante une translation géométrique de type translation, définie par le vecteur de translation ayant pour sommets (XD, YD) et (XF, YF).

  * ROTATION (DXS, DYS, DXY)
    attribue à l'identité courante une transformation géométrique de type rotation, définie par son centre de rotation de coordonnées (DXS, DYS) et l'angle de rotation (DXY) en degrés.

  * ZOOM (K)
    attribue à l'identité courante une transformation géométrique de type homothétie, définie par son coefficient (K).

- Attribution d'ECLAIRAGE :

  * ATTRIBUTECLAIRAGE (éclairage, code1, code2)
    attribue à l'identité courante un attribut d'éclairage de type "éclairage" et de valeurs "code1" et "code2".
    Les valeurs de "éclairage" sont des entiers :
    - ECL-COULEUR (400)
    - ECL-DIRECTION (401)
    - ECL-LUMAMB (402)

    Les valeurs de "code1" et "code2" sont des réels:

- > COULEUR : "code1" prend les valeurs de 0 à 4095, exprimant une couleur en (RVB), "code2" n'est pas renseigné.

- > DIRECTION : "code1" et "code2" prennent des valeurs comprises respectivement entre -pi et +pi, et entre 0 et +pi.

- > LUMAMB : "code1" prend des valeurs comprises entre 0 et 15 (LUMAMB nulle: 0, LUMAMB maxi 15), "code2" n'est pas renseigné.

- Attribution de la Géométrie de POINT DE VUE (Gv2D) :

  * FENETRE (X1, Y1, X2, Y2)
    attribue à l'identité courante une transformation géométrique de type fenêtre, définie par ses coins inférieur (X1, Y1) et supérieur (X2, Y2)

- Attribution de la Géométrie d'AFFICHAGE (Ga) :

  * CLOTURE (X1, Y1, X2, Y2)
    attribue à l'identité courante une transformation géométrique de type

cloture, définie par ses coins inférieur (X1, Y1) et supérieur (X2, Y2).

2.2.3. Les primitives de consultation

Elles sont tout à fait symétriques à celles de structuration et d'attribution

- Consultation de STRUCTURES

  * CONSULTERSTRUCTURE : fonction donnant le nom de la structure complète

  * CONSULTERCONTEXTE : fonction donnant le nom de la structure constituant le contexte

  * CONSULTERIDENTITE (var identite, pchaine) : procédure donnant le type d'identité -éphémère ou permanente- et le nom de la structure constituant l'identité

- Consultation MORPHOLOGIQUE

  * CONSULTERTYPEMORPHOLOGIQUE : fonction rendant le type morphologique -"morpho" en paragraphe 2.2.2-

  * CONSULTERVECTEUR2D : fonction rendant la liste des coordonnées des objets de type VECTEUR2D

  * CONSULTERCERCLE2D (var p1, p2, p3) : procédure rendant les coordonnées du centre et le rayon des objets de type CERCLE2D

- Consultation d'ASPECT

  * CONSULTER (Aspect, var code) : procédure rendant l'information concernant le type aspect "aspect" -cf paragraphe 2.2.2- demandé

- Consultation GEOMETRIQUE

  * CONSULTERGEOMETRIE (var mat1, mat2) : procédure rendant la matrice de transformations géométriques et son inverse. Ce point fait l'objet d'un développement en paragraphe 5.1.2 du chapitre V

- Consultation de VISUALISATION

  * CONSULTERVISUALISATION (fce; var xg,yg,xd,yd) : procédure permettant de récupèrer les coins inférieur gauche et supérieur droit de l'espace utilisateur, fenêtre ou cloture selon la valeur de "fce" :

1 Espace utilisateur
2 Fenetre
3 Cloture

2.2.4. Les primitives de visualisation

* VISUALISER (p) : visualise l'identité courante selon un processus (p). Les valeurs de (p) sont des entiers :
  .1 : ABSOLUE : visualisation par rapport au contexte global, soit la racine du fichier graphique
  .2 : RELATIVE : visualisation par rapport au contexte courant.

2.2.5. Les primitives de description

* IDENTIFIER (var p1, var chaine~car) : identifie par désignation sur l'écran, donnant en retour "p1", booléen indiquant si il y a eu identification ou non, et "chaine~car" exprimant, si "p1" vrai, la chaine de caractères exprimant le nom de la structure désignée.

* DECRIRE~COULEUR (var p1) : permet de décrire une couleur en (RVB) à partir de données dans l'espace (TLS) : Teinte ~vers le rouge, bleu ou vert~, luminance ~(+) or (~)~, Saturation ~(+) ou (~)~, donne en retour "p1", entier désignant le numéro de pointeur dans la mémoire de textures.
  Cette primitive est précédée par deux autres actions :
  .AFFICHER~PALETTE (X, Y, K) : affichage d'une palette de couleurs dans une cloture de coin inférieur gauche (X,Y) et de coefficient d'homothétie (K).
  .IDENTIFIER~COULEUR (var p1) : identifie par désignation sur l'écran, la couleur "p1" à partir de laquelle est décrite la couleur finale souhaitée par la primitive DECRIRE~COULEUR.

* DECRIRE~ECLAIRAGE (var p1, p2, p3, p4) : permet de décrire les paramètres d'éclairage de facon interactive ~(+) ou (~) par rapport à des valeurs de base~ et donne en retour les valeurs des paramètres obtenus : "p1" ~couleur de la source en RVB~, (p2, p3) ~angles donnant la direction de la source~, "p4" ~valeur de la lumière ambiante~.
  Cette primitive de description implique à chaque "pas" de description une visualisation de l'identité courante.

2.2.6. Primitives annexes

* COORDONNEES (var X, Y) : permet une collecte de coordonnées (X, Y).
* LIRENTIER (p1) : fonction booléenne vraie si "p1" est un entier.
* LIREEL (p1) : fonction booléenne vraie si "p1" est un réel.
* LIRECHAINE (p1) : fonction booléenne vraie si "p1" est une chaine de caractères.
* CHOIXTERMINAL (p1) : initialise différents paramètres, selon le matériel choisi; "p1" est un entier :
  .HELIOS : 0
  .TEKTRO 4114 : 1

2.2.7. Tableau récapitulatif des commandes

CODES COMMANDES	CODES OPTIONS	BORNES VALEURS	PRIMITIVES ASSOCIEES
I			Identite (chaine-car)
J			Identitephemere (chaine-car)
C			Contexte (chaine-car
X			Fincontexte
Y			Finidentite
S			Structures (chaine-car)
D			Detruire
E			Effacer
V			Visualiser
Z			Zoom (k)
T			Translation (xd, yd, xf, yf)
R			Rotation (dxs, dys, dxy)
M	1		Attributvecteur2d (mpoints2d, liste-coord)
	2		(msegments2d,liste-coord)
	3		(mlignes2d, liste-coord)
	4		
	4.1		Attributcercle2d (mcercle2d, p1, p2, p3)
	4.2		(mdisque2d, p1, p2, p3)
	5		Attributextefdf (mt2d, p1, p2, long-ch,p3)
	6		Attributvecteur2d (mquadri2d, liste-coord)
	7		(mface2d, liste-coord)
A	1	(1,3)	Détermination groupes : /.s classe 1. ACAR 2. ATRAIT 3. ASURF
	2		
	2.1	(1,495)	Attributaspect (/.s classe-couleur, code)
	2.2	(1,4)	(/.s classe-visibilité, code)
	2.3	(1,8)	(/.s classe-graphique, code)
	2.4		
	2.4.1	(1,4)	(/.s classe-orientation,code)
	2.4.2	(1,2)	(/.s classe-écriture, code)
	2.4.3	(1,8)	(/.s classe-taille, code)
	2.4.4	(1,2)	(/.s classe-police, code)
	2.5	(1,4)	(/.s classe-réflexion, code)
	2.6	(1,3)	(/.s classe-remplissage,code)
L	1	(0,4095)	Attributeclairage (ecl-couleur, code1)
	2	(-pi,pi) (-pi/2, pi/2)	(ecl-direction,code1,code2)
	3	(0,15)	(ecl-lumamb, code1)
U	1		Affichage-palette (xs, ys, k)
	2		Decrire-couleur (var p1)

3	Decrire-eclairage (var p1, p2, p3, p4)
4	Fenetre (x1, y1, x2, y2)
5	Cloture (x1, y1, x2, y2)
>	Identifier (var p1, var chaine-car)

Fiure IV.3: Tableau récapitulatif des commandes et primitives associées

3. Point de vue du concepteur

3.1. Gestion de la structure de données

Cette tâche a une place prépondérante dans l'aspect système du logiciel, et constitue l'essentiel de l'unité de communication.

Ce paragraphe nous permettra de rappeler et de préciser la description d'un noeud de cette structure arborescente -cf "modèlisation interne des informations" du paragraphe 3.3.1 chapitre III-, de donner les principes de base de cette gestion, et enfin d'indiquer successivement le modèle choisi pour cette gestion, les structures de données principales et les primitives associées.

3.1.1. Présentation d'un noeud de l'arborescence

Chaque noeud décrit des informations d'identité et structure d'une part, de morphologie, aspect, géométrie, éclairage d'autre part. De cette classification en 2 groupes découlent 2 types d'enregistrements au niveau du noeud.

* Enregistrement de type 'noeud' : la figure IV.4 en donne la représentation

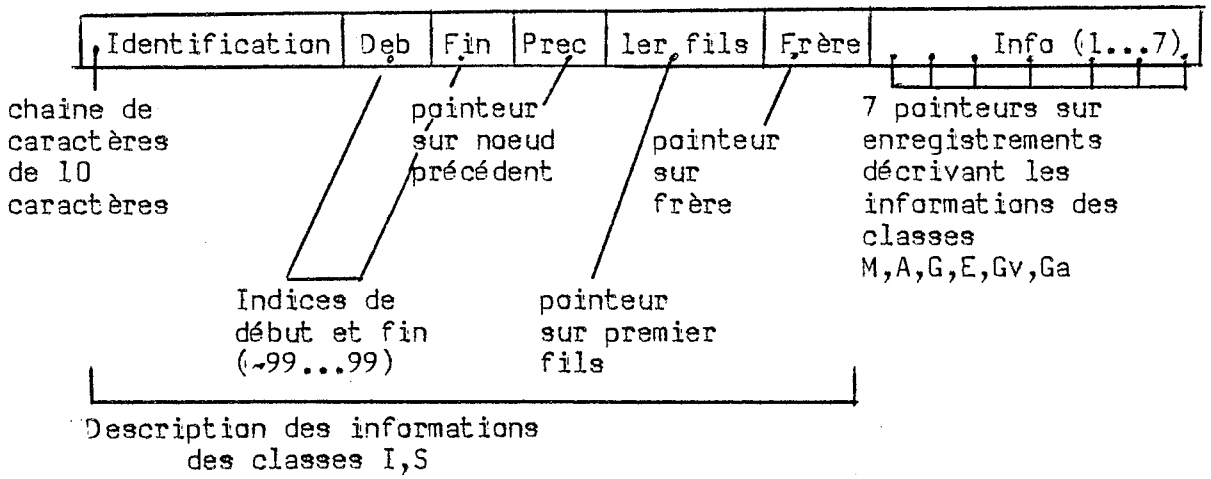


Figure IV.4: Description d'un noeud. Enregistrement de type 'NOEUD'

Il est à noter que les 7 pointeurs sur les enregistrements du second type -type "attribut"- sont tout à fait banalisés et alloués au fur et à mesure de la description des informations -processus d'attribution essentiellement-

* Enregistrement de type "attribut" : à des fins d'homogénéité de traitement des différents attributs (M, A, G...) au niveau du système, et en particulier pour ce qui est de la gestion des pointeurs, ces enregistrements ont une partie commune, décrite par la figure IV.5. La partie spécifique à chacun des enregistrements selon la classe d'information sera développée dans le chapitre suivant.

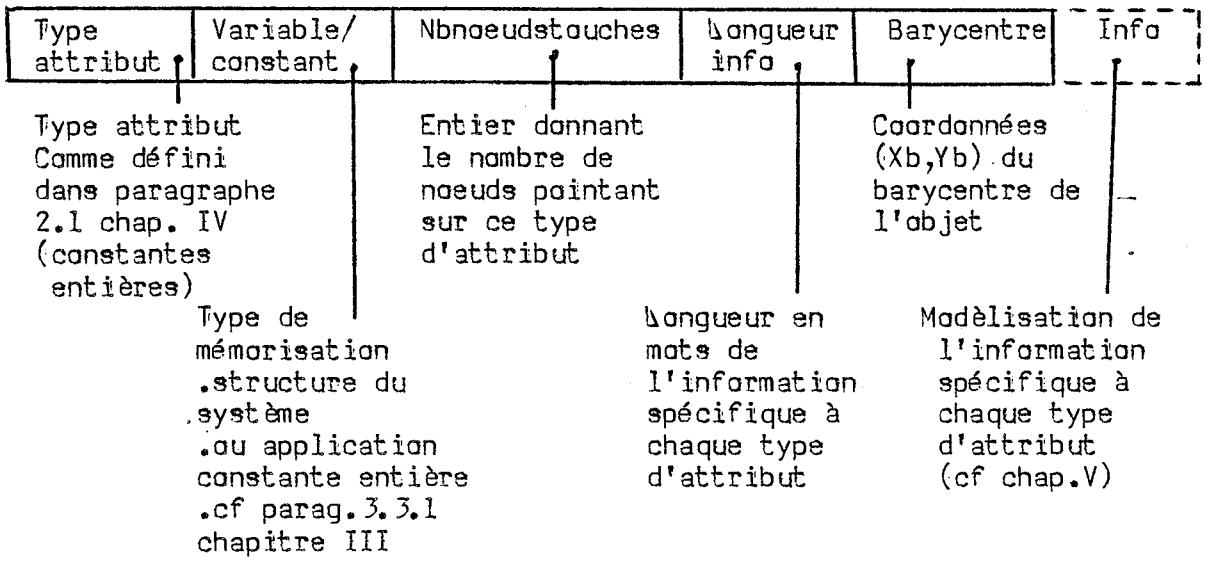


Figure IV.5: Descr. d'un noeud. Enreg. de type 'attribut' -PARTIE COMMUNE-

De ces deux descriptions, ressortent des éléments importants de gestion de la structure, sur lesquels porteront les lignes suivantes :

- Identification du noeud, et les problèmes d'analyse syntaxique en découlant
- Syntaxe d'indexation des noeuds
- Gestion du nombre de noeuds touchés
- Intérêt du calcul du barycentre

3.1.2. Principes généraux de fonctionnement

La gestion de la structure comporte deux phases principales :

- analyse de la chaine de caratères donnée en entrée
- gestion des noeuds proprement dite

* Analyse des chaines de caractères : cette analyse est faite d'une part à un niveau syntaxique -expressions, identifieurs, bornes: cf règles syntaxiques paragraphe 3.3.1 du chapitre III-, d'autre part à un niveau sémantique -2 modes: construction (essentiellement primitives 'STRUCTURES') et référence (primitives 'IDENTITE', 'CONTEXTE')-. En parallèle, est assurée la génération d'un programme de parcours de l'arbre avec empilement pour les différents noeuds des programmes concernés. Cet empilement est exploité, lors des actions portant sur les chaines définies -en particulier 'VISUALISER', ou 'AFFECTER'-, par des procédures assurant l'éxécution d'une action sur la pile de programmes ou sous-arbre issu d'un noeud. Les principales procédures mises en jeu par ces tâches autour de l'analyse seront étudiées dans le paragraphe 3.1.4 de ce même chapitre. Cette analyse fait appel à un automate décrit dans le paragraphe suivant.

* Gestion des noeuds : suite à l'analyse des chaines entrées, sont déclenchées des actions de création, éclatement ou destruction d'enregistrment de type "noeud". Ces quelques procédures seront aussi étudiées dans le paragraphe 3.1.1.

3.1.3. Automate d'analyse

Cet automate, éxécuté par la procédure 'STRUCTURE' -cf paragraphe suivant- modèlise la tâche d'analyse des expressions entrées.

Il utilise d'une part les symboles de la grammaire de ces expressions à savoir :

( ) , + - * : ! [ ]

d'autre part des symboles propres :

o/o	->	fin d'expression à analyser
n	->	identificateur
e	->	entier
?	->	caractère illégal

La figure IV.6 schématise cet automate. Il est représenté sous forme de matrice MAT (0:16, 0:12) donnant en fonction de l'état courant et du symbole courant, l'état suivant -cf figure IV.7-.

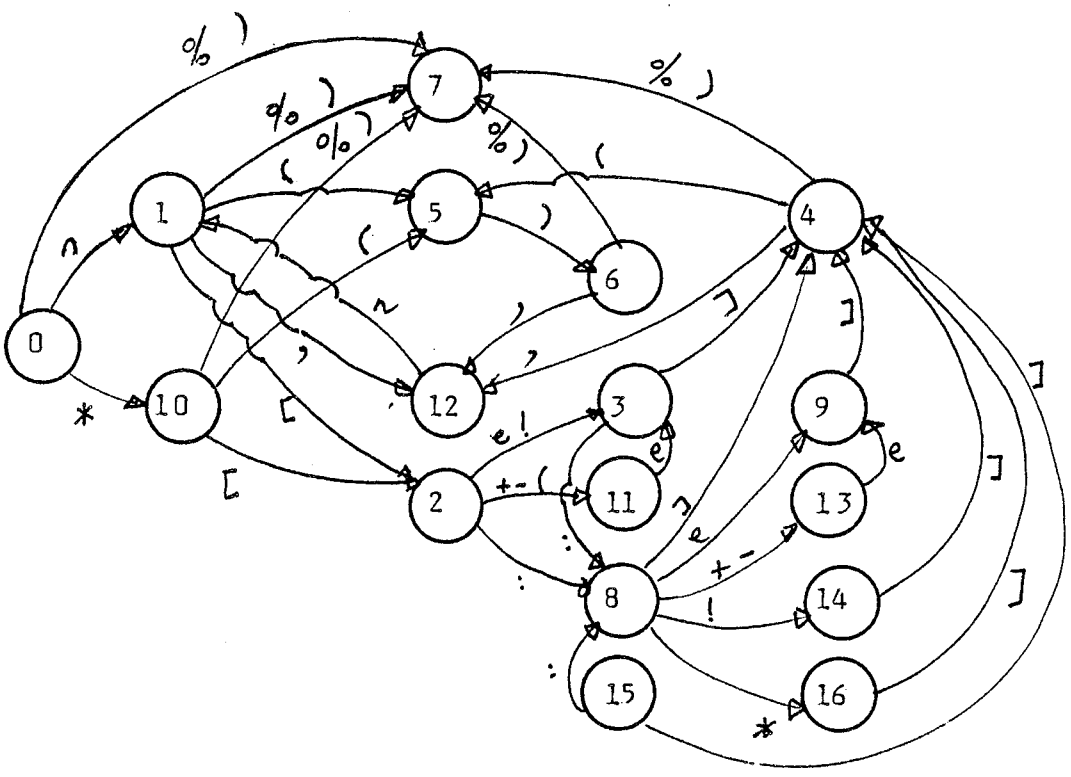


Figure IV.6: Schéma de l'automate d'analyse

Remarque : les états non valides sont codés à 20 -erreur syntaxique- dans la structure MAT représentée en figure IV.7

SYMBOLE SUIVANT (s.s) / CODE du s.s / ETAT COURANT	a/a 0	n 1	e 2	[3	] 4	, 5	+/- 6	* 7	(8	) 9	: 10	? 11	! 12
0.	7.	1.						10.		7.			
1.	7.			2.		12.			5.	7.			
2.			3.				11.	15.	11.		8.		3.
3.					4.						8.		
4.	7.					12.			5.	7.			
5.										6.			
6.	7.					12.				7.			
7.													
8.			9.		4.		13.	16.					14.
9.					4.								
10.	7.			2.					5.	7.			
11.			3.										
12.		1.											
13.			9.										
14.					4.								
15.					4.						8.		
16.					4.								

Etat suivant

Figure IV.7: Matrice de l'automate d'analyse : états valides

3.1.4. Primitives et structures de données internes associées

Nous énoncerons ici les principales procédures mises en oeuvre par la gestion de la structure graphique, ainsi que les différentes données associées.

* ANALYSE (chaine, mode; var erreur)

- Données :
  - "chaine": chaine de caractères en entrée à analyser
  - "mode": contexte d'analyse
    - 'c': création
    - 'u': chemin unique
    - 'a': absolu = à partir de la racine de la structure

."erreur": résultat contenant éventuellement le type de l'erreur
'OK': pas d'erreur
'SYNT': erreur syntaxique dans l'expression
'INCON': identificateur de noeud inconnu
'UNIQ': l'expression de référence doit être un chemin unique
'MULT': identificateur de noeud déjà existant
'BORN': erreur dans l'indice d'un noeud

- Traitement : appel de DEBPROG, STRUCTURE, si erreur appel de FINPROG

* DEBPROG (absolu)

- Données : "absolu": booléen vrai si nom absolu -cf paragraphe 3.3.1 du chapitre III-

- Traitement :
.si (absolu), empilement du pointeur de la racine de l'arbre et d'un nouveau niveau de programme
.sinon, empilement d'un niveau de programme uniquement,16 niveaux étant possibles.

* FINPROG : dépile le programme courant et restaure la pile des noeuds à l'état primitif.

* STRUCTURE (i; var erreur)

- Données :
."i": indice de la chaine à traiter
."erreur": type d'erreur détectée

- Traitement : procédure récursive assurant l'éxécution de l'automate
Génère en parallèle le programme de parcours : "GENERE"
Assure la gestion des noeuds : éclatement, création...: "ECLATENOEUD", "INSERERNOEUD", "DETRUIRENOEUD".

* GENERE (code, adress)

- Données : ."code": commande de parcours d'une structure :
'FD': feuilles en descendant
'FA': feuilles en ascendant
'BD': branches en descendant
'BA': branches en ascendant
'PN': à partir de la racine de l'arbre
."adresse": adresse du noeud

- Traitement : génère l'instruction (code, adresse) et l'empile dans CODECDE, indicée par la pile du programme courant FINP, elle-même indicée par le numéro de niveau -cf figure IV.8-. L'adresse est empilée dans ADRCDE, indicée par FINP -cf figure IV.9-.

Provoque l'empilage et le dépilage des noeuds parcourus: "EMPILE", "DEPILE".

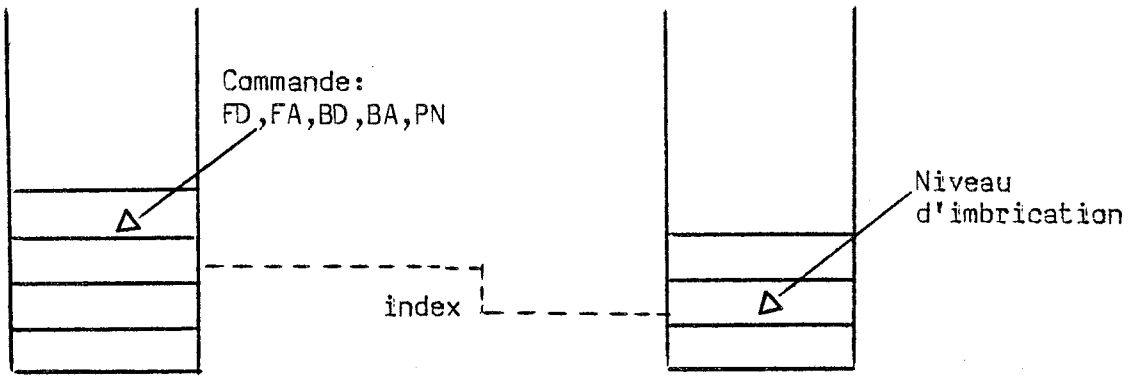


Figure IV.8: Piles systèmes : génération du parcours de l'arbre

* ECLATENOEUD (orig, d1, f1, d2, f2)

- Données :
  - ."orig": adresse du noeud à éclater
  - ."d1", "f1": début et fin de l'ancien noeud
  - ."d2", "f2": début et fin du nouveau noeud
- Traitement : génère la création du sous-arbre attaché au noeud

* INSERERNOEUD

- Données :
  - ."p": adresse du noeud à insèrer
  - ."pere": adresse du père du noeud à insèrer
- Traitement : le nouveau fils est insèré à sa place dans l'ordre alphabétique.
  La pile PILE, des noeuds courants, est mise à jour avec l'adresse du nouveau noeud, -cf figure IV.9-

* DETRUIRENOEUD

- Données : ."p1": adresse du noeud à détruire

- Traitement : détruit le noeud pointé par "p" et les attributs associés.

  * EMPILE (adr)

- Données : ."adr": adresse du nouveau noeud courant

- Traitement : empile dans PILE, les informations suivantes : adresse du noeud courant "adr", borne inférieure d'indexation du noeud courant "ind", index sur le chemin du parcours "icour" déterminé par le programme de parcours ayant provoqué l'empilement du noeud. Le premier élément de PILE contient la racine de l'arbre complet -cf figure IV.9-.

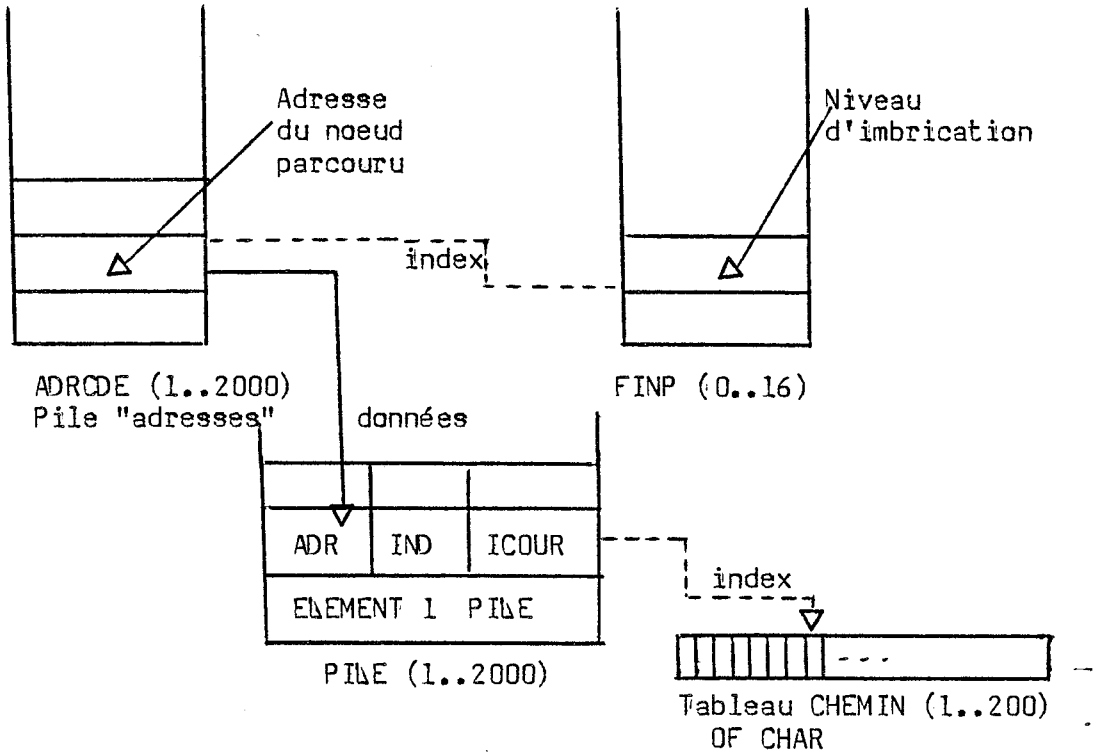


Figure IV.9: Piles systèmes. Gestion des pointeurs d'adresses

  * EXECARBRE (adr, racine, mode, procedure)

- Données :

  ."adr": pointeur sur la racine de l'arbre à traiter
  ."racine": booléen vrai si le traitement inclut la racine
  ."mode": sous-ensemble de (FD,FA,BD,BA,PN)
  ."procedure": nom de la procédure à éxécuter

- Traitement : éxécute la procédure "procedure" lors du parcours préfixé ou postfixé -selon la valeur de "mode"- de l'arbre dont la racine a pour adresse "adr"

* EXECPROG (nprog, racine, mode, procedure)

- Données :
  ."nprog": niveau de programme à traiter
  ."racine": booléen vrai si le traitement inclut la racine de l'arbre
  ."mode": sous-ensemble de (FD,FA,BD,BA,PN)
  ."procedure": nom de la procédure à éxécuter

- Traitement : éxécute la procédure de nom "procedure" pour le programme empilé courant pointé par "nprog".

3.1.5. Gestion du nombre de noeuds touchés

Chaque objet d'une scène (ou noeud de l'arborescence) peut être complèté par un ou plusieurs types d'attributs.

Plusieurs noeuds peuvent partager les mêmes attributs, de même qu'un même noeud, selon l'indice qu'il porte, peut avoir des attributs différents.

De ces constatations, résulte que chaque enregistrement de type "attribut" possède un compteur de référence "NBNOEUDSTOUCHES" -cf figure IV.5-, permettant de détecter un attribut non référencé -NBNOEUDSTOUCHES = 0- et ainsi de le détruire.

Le calcul de ce compteur, via la procédure "NBNOEUD" intervient lors de l'affectation d'un attribut à un ou plusieurs noeuds, et lors de le destruction d'un ou plusieurs noeuds.

* NBNOEUDS (z, pp, nbnoeudstouches) :

- Données :
  ."z": nombre de noeuds concernés par le calcul
  ."pp": pointeur sur l'un des enregistrements de type "attribut"
  ."nbnoeudstouches": résultat du calcul

- Traitement : calcul: nbnoeudstouches - z. Si ce calcul donne un résultat nul,l'enregistrement du type considèré est libèré, sinon "nbnoeudstouches" est calculé dans l'enregistrement.

3.1.6. Calcul du barycentre

Ce calcul s'applique aux enregistrements de type morphologique, lors de la définition de l'attribut morphologique et lors de toute transformation géométrique subie par l'objet.

Cette notion de barycentre est très importante lorsqu'une transformation géométrique porte sur un ensemble d'objets, chacun des barycentres des différents objets permettant un "centrage" correct de la transformation globale. Les coordonnées (Xb,Yb) mémorisées dans l'enregistrement de type morphologique -cf figure IV.5- sont calculées simplement :

$$Xb = \Sigma Xi / i$$
$$Yb = \Sigma Yi / i$$

avec i : nombre de points définissant l'objet
(Xi,Yi) : coordonnées de ces points

3.1.7. Conclusion

Ce noyau du logiciel a été testé tant en interactif, par le développement d'exemples concernant les divers matériels du laboratoire -HELIOS, TEKTRONIX-, qu'à partir d'un programme d'application, en particulier grâce aux travaux de J.C. MARTY, auteur d'une thèse de 3eme cycle (MAT84) dans laquelle il développe un éditeur graphique, dans le cadre du projet CASCADE réalisé par l'équipe CAO du laboratoire ARTEMIS, et où il utilise comme logiciel de base le logiciel présenté ici.

3.2. Gestion des processus

Cette tâche, incombant à l'unité de controle, est, nous l'avons vu -cf paragraphe 3.3.2 du chapitre III- une des particularités du logiciel présenté ici, et concerne, rappelons-le encore, quatre procesus de base :

- ATTRIBUER
- CONSULTER
- VISUALISER
- DECRIRE

Cette gestion consiste en un ordonnancement des processus en fonction du choix de ceux-ci, conditionné par différents facteurs. C'est par ce point que nous commencons cette étude de la gestion des processus. Le deuxième point d'étude s'attache à donner quelques éléments de travail sur la mémorisation des informations, qui est un des aspects sur cet équilibrage des tâches entre les différentes couches logicielles que nous avions déjà évoqué -cf paragraphe 1.2 de ce chapitre-. Nous terminerons par l'énoncé des processus choisis ainsi que des structures de données aidant ce choix.

3.2.1. Facteurs influant la gestion des processus

Il est clair que, par ce travail d'ordonnancement, l'unité de contrôle du logiciel présenté est responsable de l'adéquation recherchée entre les différents matériels et applications.

Ainsi se dégagent les deux premiers facteurs d'influence :

- le type de matériel avec ses possibilités et ses vocations essentielles.

- le type d'application avec ses performances requises, telles que nous les avions définies dans le paragraphe 3 du chapitre II.

Dans le cas d'HELIOS comme matériel, nous nous trouvons dans une configuration multipliant les processus possibles. Les possibilités cablées ou micro-programmées de ce matériel sont nombreuses -calcul d'éclairage, gestion des textures, mémorisation d'attributs, élément de base évolué ("FACE")- et permettant nombre d'ordonnancements, ce que nous verrons plus en détail dans le paragraphe suivant.

En ce qui concerne le rôle de l'application, il est évident que ses exigences -nécessité d'un grand réalisme dans les images, importance du temps réel- entrainent des actions bien différentes et une exploitation spécifique des capacités du matériel associé.

Le troisième facteur influant est le type d'objet, selon surtout, pour être plus précis, la sous-classe morphologique auquel il appartient, telle que nous l'avons décrite en paragraphe 2.1.1 de ce chapitre. Ceci est très sensible pour ce qui est des processus de visualisation et de description, un élément de type "SURFACE" étant plus complexe à manipuler et incluant d'autres actions portant sur de plus nombreux attributs qu'un élément "FILDEFER" par exemple -cf figure IV.2-

3.2.2. Mémorisation des informations

Ce problème se pose en fait à plusieurs niveaux ou "frontières" entre couches logicielles :

* Logiciel d'application / Logiciel principal -Application / Système-: c'est le problème des attributs constants et variables, les premiers étant mémorisés dans la structure arborescente du système alors que les seconds le sont au niveau de l'application -cf figure III.20-.

* Au sein de la structure du logiciel principal : cette difficulté découle essentiellement de la question de la composition des attributs.
  L'exemple le plus typique reste le cas des transformations géométriques.
  L'attribution à un noeud d'une telle transformation est bien sûr mémorisée dans la structure dans l'enregistrement de type "attribut" relatif à la classe "géométrie". Mais lors du parcours de l'arbre pour visualisation, la composition des différents attributs géométriques

s'impose. Le choix réside entre calculer et mémoriser les transformations composées au niveau de chaque noeud -mais à partir de quelle racine -, ou calculer la composition au moment de la visualisation, ce choix se répercutant sur les processus d'attribution et visualisation, en tout premier lieu.

* Logiciel principal / Logiciel pilote -Système / Matériel- : Les possibilités de mémorisation du terminal HELIOS -ce qui n'est pas le cas par exemple du TEKTRONIX 4114- impliquent ce choix. On sait que HELIOS comporte une structure de données comprenant des informations telles que couleur ou texture, visibilité, modèle de réflexion, correspondant toutes à des informations d'aspect impliquées dans le processus d'attribution. La spécificité essentielle réside dans le lien de ces attributs à un identifieur, qui est la face, ce qui entraine une gestion d'un lien entre les éléments manipulés par la structure et les faces manipulées par le matériel. A partir de là, deux processus d'attribution et visualisation sont possibles : mémorisation au niveau de la structure des numéros de faces HELIOS, au niveau matériel des attributs et communication lors de la visualisation des seuls identifieurs de faces, ou bien mémorisation dans la structure des attributs et communication à chaque visualisation des paramètres valides.

Le paragraphe suivant propose des choix et des solutions associées quant à ces problèmes.

3.2.3. Solutions retenues

La réalisation de ce logiciel pilote d'HELIOS a été guidée, comme toute première réalisation, par une évolution vers une complexité croissante. Deux restrictions s'imposent ici :

- la réalisation n'a concerné que des éléments 2D

- un problème comme l'élimination des parties cachées n'a pas été traité, ne soulevant pas ainsi les questions concernant la détermination du point de convergence entre des processus utilisant des éléments différents -points, segments, faces- selon les algorithmes de visibilité choisis - Z-buffer, Watkins, Newell-Newell-Sancha -

Toutefois, la réalisation, même si elle n'a pas effectivement apporté de solutions, a tenu compte de ces problèmes par des structures de données et des traitements adéquats. Nous pouvons dire que 2 types de processus ont été envisagés, fonctions de l'aspect mémorisation des attributs :

- mémorisation dans la structure du système

- mémorisation dans la structure du matériel

* Processus 1: mémorisation "système" : ce choix nous est apparu indispensable en tant que solution la plus immédiate et surtout permettant de valider la gestion de la structure arborescente. Nous indiquons ci-après la composition des processus ATTRIBUER et

VISUALISER -comme nous le ferons pour le processus 2-, non sans avoir rappelé au préalable les opérateurs dont dispose l'unité de controle -cf paragraphe 3.3.2 du chapitre III-

- opérateurs de l'unité de communication, essentiellement STRUCTURE, IDENTITE, CONTEXTE, PARCOURS, AFFECTER

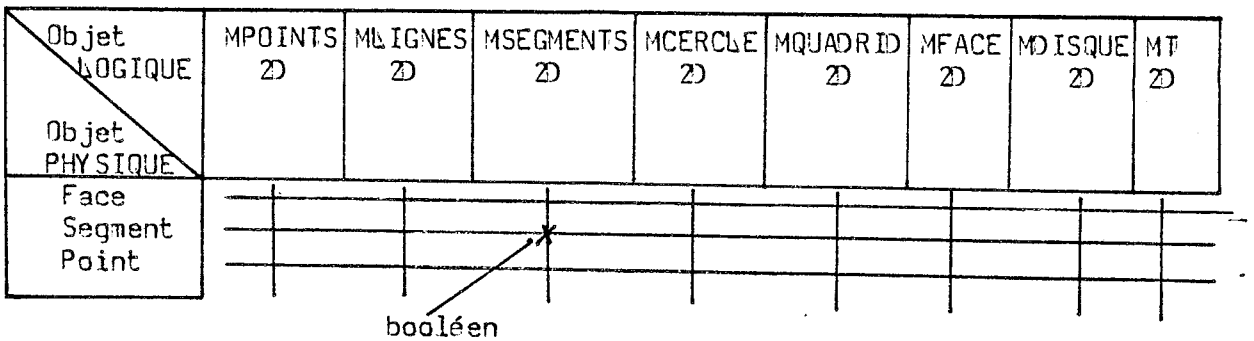
- opérateurs du logiciel pilote, ATTRIBUER-pilote, VISUALISER-pilote essentiellement :

  . Processus ATTRIBUER :
  IDENTITE (structure)
  PARCOURS (structure)
  AFFECTER (noeud, attributs)
  Marquage des noeuds

  . Processus VISUALISER :
  IDENTITE (structure)
  PARCOURS (structure)
  Decomposition-objet (éléments-base)
  ATTRIBUER-pilote (éléments-base, attributs)
  VISUALISER-pilote

La phase "Décomposition-objet" nécessite deux structures de données :

- Structure 1 : tableau indiquant pour chaque objet (objet "LOGIQUE") si il peut se décomposer ou non en certains éléments de base (objet "PHYSIQUE"). Cette structure -cf figure IV.10- est initialisée dans la primitive "CHOIXTERMINAL" en fonction du terminal et du processus.




```
Initialisation : STRUCTURE1 (*, FACE):=TRUE;
                 STRUCTURE1 (*, SEGMENT):=FALSE;
                 STRUCTURE1 (*, POINT):=FALSE;
```

Figure IV.10: Lien objets logique-physique

- Structure 2 : tableau indiquant pour chaque objet "physique" identifié par un entier si il a déjà été attribué ou non à un objet "logique". HELIOS comportant 1024 faces possibles, il s'agit d'un tableau STRUCTURE2 (0..1023).

Dans ce processus 1, somme toute assez primaire, la visualisation déclenche la recherche d'un élément de base libre effectuée par "Décomposition-objet". Cette primitive est imposée par l'attribution obligatoire dans le cas d'HELIOS d'un identificateur de face.

* Processus 2 : mémorisation "matériel" : ce choix nous permet de mieux exploiter les possibilités d'HELIOS et éviter ainsi une certaine redondance dans la mémorisation des informations.

  . Processus ATTRIBUER :
  IDENTITE (structure)
  PARCOURS (structure)
  AFFECTER (noeud, attributs-specifiques)
  ATTRIBUER-pilote (attributs-matériel)
  Decomposition-objet (elements-base)
  AFFECTER (noeud, elements-base)
  ATTRIBUER-pilote (elements-base)
  Marquage noeuds

  . Processus VISUALISER :
  IDENTITE (structure)
  PARCOURS (structure)
  VISUALISER-pilote

Les différences essentielles par rapport au processus 1, sont les suivantes :

- mémorisation dans la structure des attributs non mémorisés par le matériel

- la primitive "Decomposition-objet" nécessite non seulement les structures citées dans le cas du processus 1, mais implique aussi la gestion dans le structure arborescente d'une nouvelle classe d'informations que nous nommerons "MATERIEL", contenant l'identification des éléments de base. Cette gestion sera développée largement dans le chapitre suivant. La communication au matériel de cette identification permet à celui-ci de consulter et rechercher les attributs concernés.

4. Conclusion

Ce chapitre nous a permis de dégager d'une part une classification des informations manipulées dans le logiciel, d'autre part une série de problèmes touchant à l'équilibrage des tâches.

Le chapitre V présente la réalisation du logiciel respectant contraintes et problèmes que nous avons eu l'occasion de soulever, dont nous

faisons ici une rapide synthèse :

- restriction du logiciel en 2D

- examen de deux types de processus selon le niveau de mémorisation des informations, et, cela en découle, le niveau d'éxécution des actions -logiciel principal, pilote et même matériel au sens strict-

- obligation d'une structuration du logiciel répondant à plusieurs impératifs :
  . "logique": respect du découpage hiérarchique en unités de controle, communication et description-visualisation
  . "physique": adaptation du logiciel à deux classes de matériel (HELIOS, et TEKTRO et compatibles)
  . modularité pour des extensions tant pour l'utilisateur que le concepteur.

CHAPITRE V : REALISATION LOGICIELLE
ASPECT TECHNIQUE

Il s'agit là du dernier chapitre de ce mémoire, en cloturant la démarche que nous avons voulue évolutive et progressive, allant du plus général au plus précis. La réalisation du logiciel y est donc présentée sur un plan strictement technique, dont les points forts sont les suivants :

- présentation des matériels utilisés
- présentation du logiciel en termes de description de données, codage d'informations et structuration en modules
- différentes réalisations classées selon l'unité dans laquelle elles sont mises en oeuvre: contrôle, communication, et description-visualisation

Nous terminerons par un exemple d'utilisation de ce logiciel, à la fois donc validation et illustration de ce travail.

1. Présentation des matériels

1.1. Le calculateur principal

Comme nous l'avons déjà indiqué, il s'agit d'un VAX 11/780 de DEC, appartenant à la société MICADO située dans la ZIRST de Meylan.

Ce matériel est le plus puissant de la famille des DEC-11, à laquelle appartiennent aussi les séries LSI-11 et PDP-11.

Voici rapidement ses principales caractéristiques:

- mots de 32 bits
- système d'exploitation gérant une mémoire virtuelle : VAX/VMS -Virtual Adress eXtension/Virtual Memory System-
- unité centrale d'une capacité maximale de 2 Millions d'octets
- mémoire de masse de 4.3 Billions d'octets, d'où un espace virtuel adressable de la même taille -2**32 octets-
- pages de 512 octets
- 32 niveaux de priorité: 0 à 15 pour les processus normaux, 16 à 31 pour les processus nécessitant des temps de réponse plus performants.

Le VAX 11/780 (VMS) propose une gamme d'outils de développement

classiques:

- Editeur de texte -EDT- ligne -LINE editor- ou pleine page -KEYPAD Editing-

- Editeur de liens -LINK-

- Compilateur PASCAL -VAX-11 PASCAL- acceptant de nombreuses extensions: déclaration de chaines de caractères (VARYING of CHAR), insertion de modules sources (INCLUDE), instructions particulières -clause OTHERWISE dans un CASE-, déclaration d'en-tête de procédures séparément du corps -FORWARD-, ...

- Manipulation de librairies de programmes objets -LIBRARY-

- Gestion d'une ou plusieurs machines par utilisateur, avec mots de passe et directories internes.

En résumé, un matériel dans la ligne des 32 bits, multi-utilisateurs, et satisfaisant des applications commerciales ou scientifiques.

1.2. Le calculateur satellite

Il s'agit là de la PASCALINE MICROENGINE, microordinateur de WESTERN DIGITAL -WD/90-. Ses principales caractèristiques sont les suivantes:

- processeur WD/9000 à mots de 16 bits

- système PASCAL-UCSD, interprétant directement le langage intermédiaire P-code

- 64 K octets de RAM

- 2 ports serie asynchrone RS232, à vitesse programmable -110 à 19200 bauds-

- 2 ports parallèles à 8 bits

- un controleur de FLOPPY, programmable pour disques 5.1/4 ou 8 pouces, simple ou double densité

- un compilateur PASCAL, permettant les extensions UCSD

- tous les outils du système UCSD, éditeur pleine page -EDIT-, éditeur de liens -LINKER-, un gestionnaire de fichiers -FILER-, ...

Ce matériel agréable d'utilisation et performant -grâce au P-code notamment- a facilité la mise au point du module d'interface implanté sur ce calculateur.

2. Présentation du logiciel

2.1. Description des données

Nous avons vu dans le chapitre précédent -cf. paragraphe 3.1.1. chapitre IV- comment se présente chacun des noeuds de la structure de données graphiques arborescente.

Deux types d'enregistrements s'en dégagent :

- type 'noeud', regroupant les informations des classes I et S

- type 'attribut', regroupant les autres classes et dont a été présentée la partie commune.

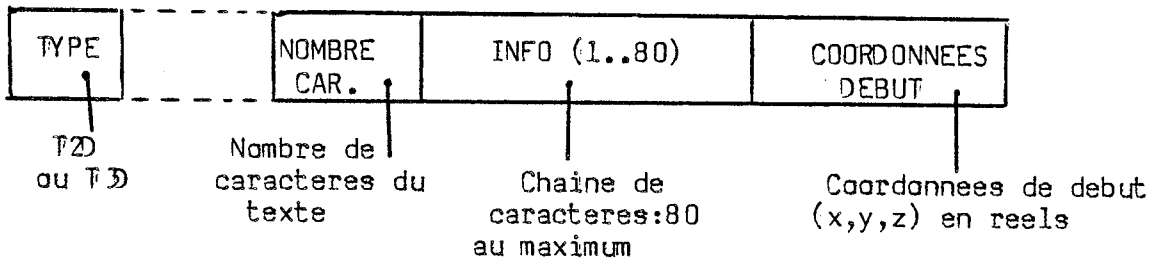
C'est la partie spécifique de ce dernier enregistrement qui va être développée dans ce paragraphe, suivie du codage des différents types d'attributs.

2.1.1. Les descripteurs d'attributs

Pour chacune des classes d'informations et d'une facon plus précise des types d'attributs, nous indiquerons un schéma de l'enregistrement suivi d'une description exprimée dans le langage PASCAL.

* MORPHOLOGIE

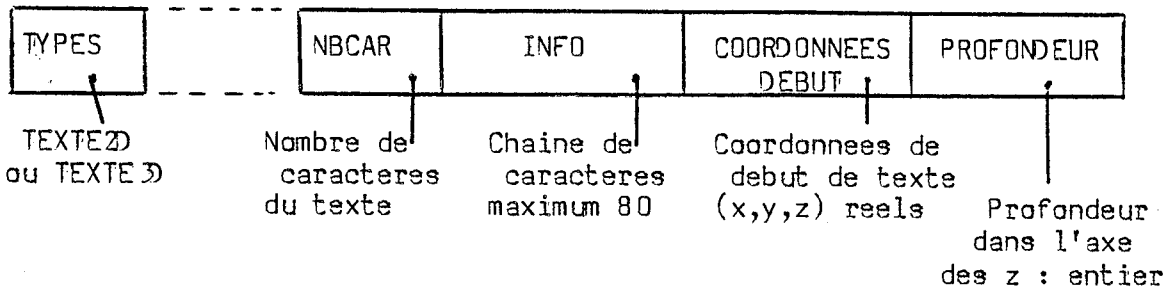
- Type "TEXTEFDF" regroupant les éléments T2D et T3D.




```
DMTEXTEFDF = record
      TYPES : TEXTEFDF;
      VARICTE : integer;
      NBNOEUDSTOUCHES : integer;
      NBCAR : integer;
      INFO : CH80;
      COORDONNEESDEBUT : array(1..3) of real;
end;
```

Figure V.1: Descripteur type attribut TEXTEFDF, classe M

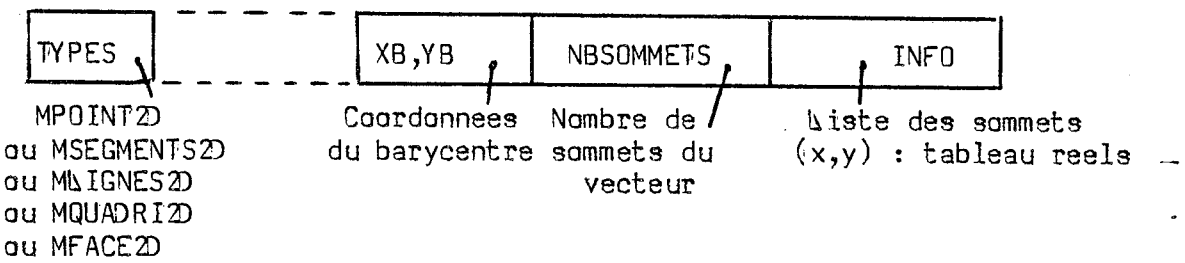
- Type "TEXTESURF" : élements TEXTE2D et TEXTE3D




```
DMTEXTESURF = record
      TYPES : TEXTESURF;
      VARICTE : integer;
      NBNOEUDSTOUCHES : integer;
      NBCAR : integer;
      INFO : CH80;
      COORDONNEESDEB : array (1..3) of real;
      PROFONDEUR : integer;
end;
```

Figure V.2: Descripteur type attribut TEXTESURF, classe M

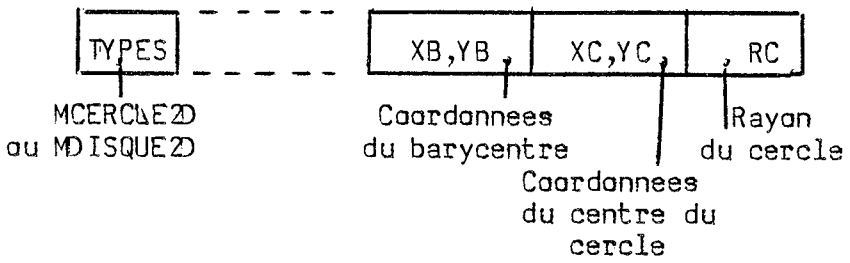
- Type "VECTEUR2D" : éléments MPOINTS2D, MSEGMENTS2D, MLIGNES2D, MQUADRI2D, MFACE2D.




```
DMVECTEUR2D = record
      TYPES : VECTEUR2D;
      VARICTE : integer;
      NBNOEUDSTOUCHES : integer;
      XB,YB : real;
      NBSOMMETS : integer;
      INFO : TABLEREELS;
end;
```

Figure V.3: Descripteur type attribut VECTEUR2D, classe M

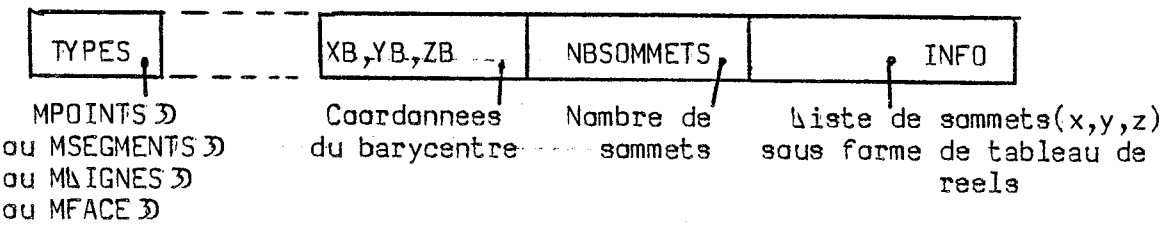
- Type "CERCLE2D" : éléments MCERCLE2D ET MDISQUE2D




```
DMCERCLE2D = record
     TYPES : CERCLE2D;
     VARICTE : integer;
     NBNOEUDSTOUCHES : integer;
     XB,YB : real;
     XC,YC : real;
     RC : real;
end;
```

Figure V.4: Descripteur type attribut CERCLE2D, classe M

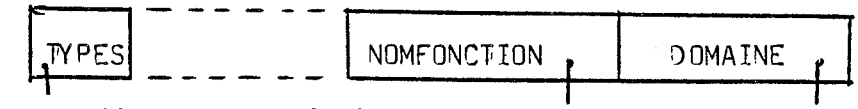
- Type "VECTEUR3D" : éléments MPOINTS3D, MSEGMENTS3D, MLIGNES3D, MFACE3D




```
DMVECTEUR3D = record
     TYPES : VECTEUR3D;
     VARICTE : integer;
     NBNOEUDSTOUCHES : integer;
     XB,YB,ZB : real;
     NBSOMMETS : integer;
     INFO : TABLEREELS;
end;
```

Figure V.5: Descripteur type attribut VECTEUR3D, classe M

- Type "FONCTION3D" : élément MFONCTION3D. Ce descripteur ne fait que proposer un type de description de fonction 3D. Il est évident que bien d'autres possibilités sont envisageables, non abordées dans le cadre de ce travail, consacré exclusivement dans un premier temps au 2D.



MFONCTION3D

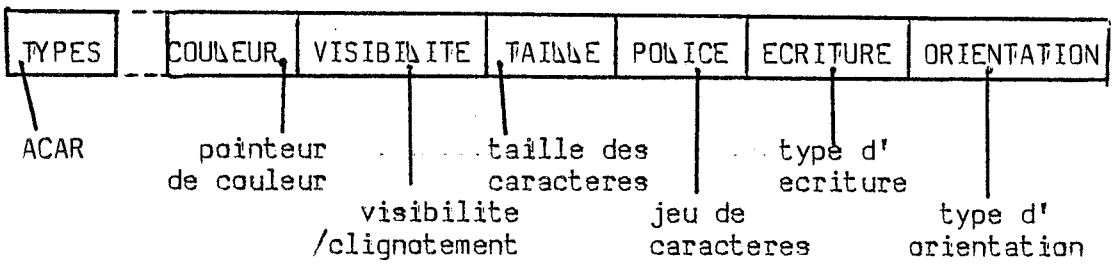
Chaine de caracteres (80 maxi.) exprimant le nom de la fonction

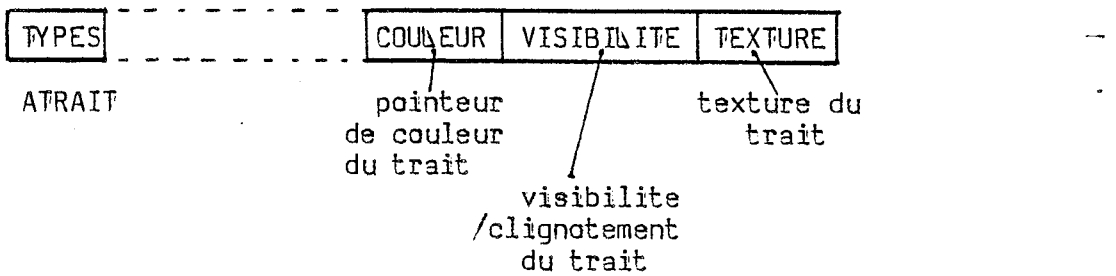
Domaine de la fonction (xdeb,xfin),(ydeb,yfin) (zdeb,zfin)

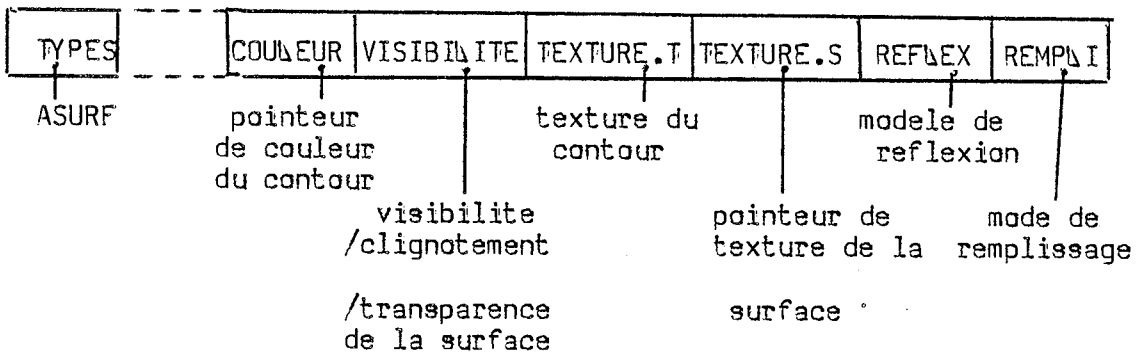
```
DMFONCTION3D = record
     TYPES : FONCTION3D;
     VARICTE : integer;
     NBNOEUDSTOUCHES : integer;
     NOMFONCTION : CH80;
     DOMAINE : TABLEREELS;
end;
```

Figure V.6: Descripteur type attribut FONCTION3D, classe M

* ASPECT : cette classe comporte trois types d'attributs : ACAR, ATRAIT, ASURF





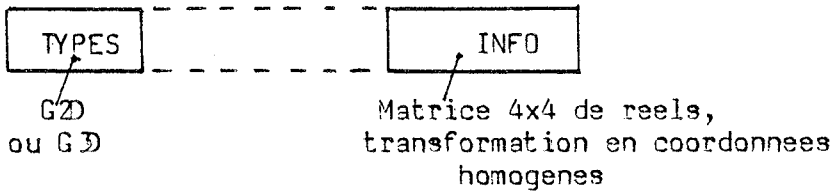


A ces trois types d'attributs, correspond un unique descripteur, contenant dans sa descripton la sélection sur les informations adéquates selon le type d'attribut considéré.

```
DASPECT = record
     TYPES : ASPECT;
     VARICTE : integer;
     NBNOEUDSTOUCHES : integer;
     COULEUR : integer;
     VISIBILITE : integer;
     case TYPES of
     ACAR : (TAILLE,POLICE,ECRITURE,
               ORIENTATION:integer);
     ATRAIT : (TEXTURETRAIT:integer);
     ASURF : (TEXTURECONTOUR,TEXTURE,REFLEXION,
              REMPLISSAGE:integer);
     end;
end;
```

Figure V.7: Descripteur types attributs ACAR, ATRAIT et ASURF, classe A

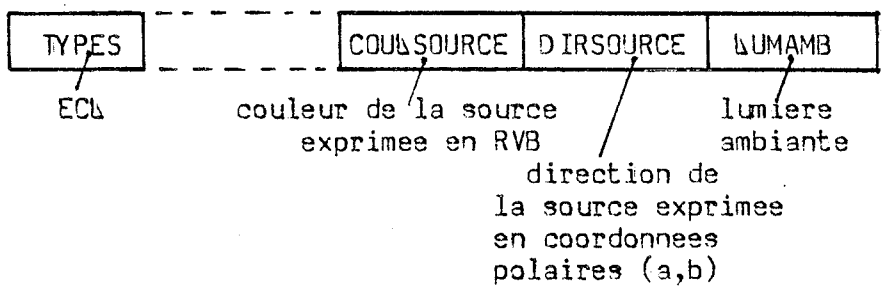
* GEOMETRIE : cette classe comporte deux types d'attributs G2D et G3D, modèlisés par un unique descripteur.




```
DGEOMETRIE = record
      TYPES : GEOMETRIE;
      VARICTE : integer;
      NBNOEUDSTOUCHES : integer;
      INFO : array(1..4,1..4) of real;
end;
```

Figure V.8: Descripteur types attribut G2D et G3d, classe G

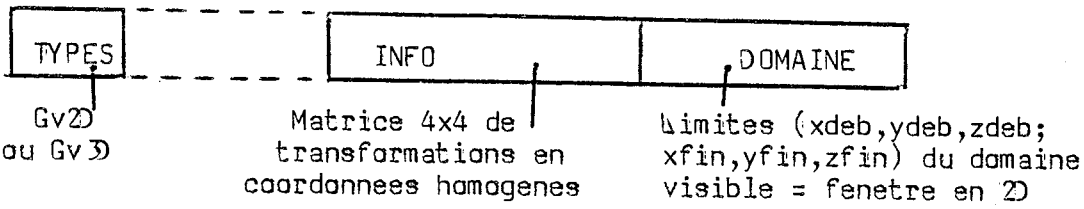
* ECLAIRAGE : un type d'attribut unique ECL




```
DECLAIRAGE = record
      TYPES : ECLAIRAGE;
      VARICTE : integer;
      NBNOEUDSTOUCHES : integer;
      COULSOURCE : integer;
      DIRSOURCE : array (1..2) of real;
      LUMAMB : integer;
end;
```

Figure V.9: Description type attribut ECL, classe E

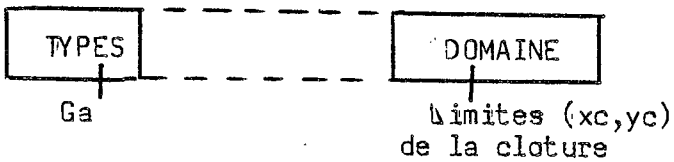
* GEOMETRIE DE PRISE DE VUE : deux types d'attributs Gv2D et Gv3D modèlisés par un descripteur unique




```
DGEOMV = record
   TYPES : GV;
   VARICTE : integer;
   NBNOEUDSTOUCHES : integer;
   INFO : array (1..4,1..4) of real;
   DOMAINE : TABLEREELS;
END;
```

Figure V.10: Descripteur types attribut Gv2D et Gv3D, classe Gv

* GEOMETRIE D'AFFICHAGE : un type d'attribut Ga, de la classe Ga




```
DGEOMA = record
   TYPES : GA;
   VARICTE : integer;
   NBNOEUDSTOUCHES : integer;
   DOMAINE : array (1..2) of real;
end;
```

Figure V.11: Descripteur type attribut Ga, classe Ga

* MATERIEL : nous avons évoqué dans le chapitre précédent -cf. paragraphe 3.2.2.- la nécessité de la gestion d'un lien entre l'objet manipulé dans la structure du logiciel et l'élément de base du matériel HELIOS, à savoir la face. C'est le descripteur que nous présentons maintenant qui permet ce lien, à l'intérieur d'une pseudo classe MATERIEL, comprenant différents types d'attributs tels MTFACE, mais aussi dans le même ordre d'idées, MTSEGMENT ou MTPOINT ...

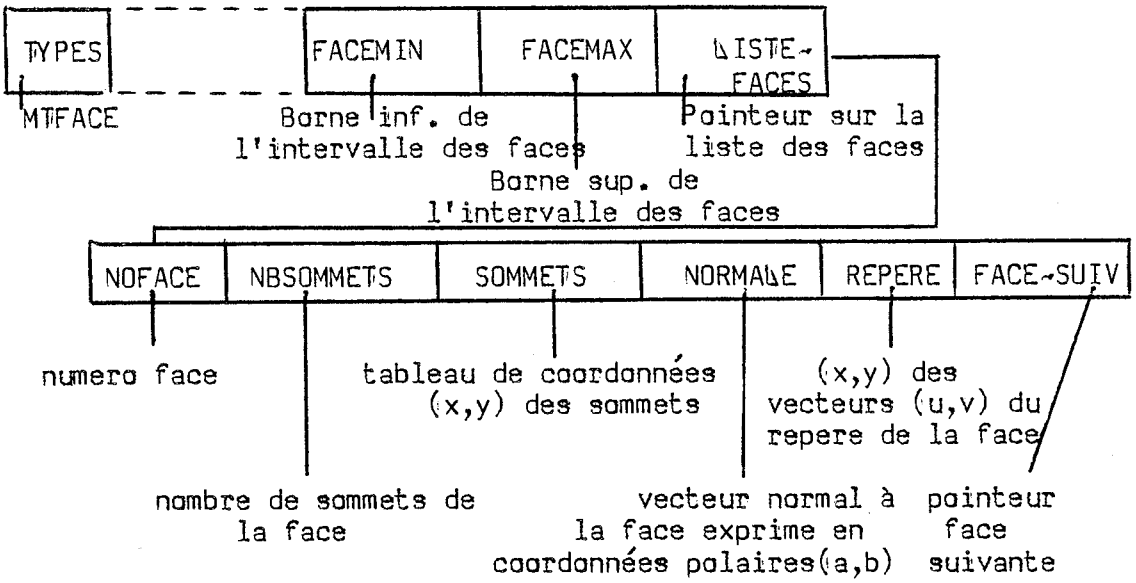


Figure V.12: Descripteur type attribut MTFACE, classe MATERIEL

Ces informations de cette pseudo-classe MATERIEL contiennent en fait des informations de la classe I -numéro de face- et de la classe E -normale et repère de la face-.

```
DMTFACE = record
    TYPES : MTFACE;
    VARICTE : integer;
    NBNOEUDSTOUCHES : integer;
    FACEMIN,FACEMAX : integer;
    INFO : PLISTEFACE;
end;
PLISTEFACE = LISTEFACE;
LISTEFACE = record
    NOFACE : integer;
    NBSOMMETS : integer;
    SOMMETS : array (1..10,1..2) of integer;
    NORMALE : array (1..2) of real;
    REPERE : array (1..2,1..2) of integer;
    SUIVANTE : PLISTEFACE;
end;
```

2.1.2. Les types d'attributs et leur codage

Classes, types et informations ont été codés sous forme de constantes entières. La figure V.13 récapitule la liste des éléments utilisés par le logiciel et leur codage.

CLASSES	TYPES	SOUS-TYPES /INFORMATIONS	CODAGE
M			7X..80
	TEXTEFDF		70..71
	TF2D		70
	TF3D		71
	TEXTESURF		72..73
	TEXTE2D		72
	TEXTE3D		73
	VECTEUR2D		74..78
	MPOINTS2D		74
	MSEGMENTS2D		75
	MLIGNES2D		76
	MQUADRI2D		77
	MFACE2D		78
	CERCLE2D		79..80
	MCERCLE2D		79
	MDISQUE2D		80
	VECTEUR3D		81..84
	MPOINTS3D		81
	MSEGMENTS3D		82
	MLIGNES3D		83
	MFACE2D		84
	FONCTION3D		85..85
	MFONCTION3D		85
A			1XX
	ACAR		10X
		ACAR.COULEUR	100
		ACAR.VISIBILITE	101
		ACAR.TAILLE	102
		ACAR.POLICE	103
		ACAR.ECRITURE	104
		ACAR.ORIENTATION	105
	ATRAIT		11X
		ATRAIT.COULEUR	110
		ATRAIT.VISIBILITE	111

		ATRAIT.TEXTURE	112
	ASURF		12X
		ASURF.COULEUR	120
		ASURF.VISIBILITE	121
		ASURF.TEXTURECONTOUR	122
		ASURF.TEXTURE	123
		ASURF.REFLEXION	124
		ASURF.REMPLISSAGE	125
G			3X
	G2D		30
	G3D		31
E			4X
	ECL		40
Gv			2X
	Gv2D		20
	Gv3D		21
Ga			5X
	Ga		50
MT			6X
	MTFACE		60
	MTSEGMENT		61
	MTPOINT		62

Figure V.13: Classification et codage des informations

2.2. Description des modules

Les modules réalisés dans le cadre de ce logiciel, outre l'organisation hiérarchique en trois unités, peuvent se classer en trois groupes :

- interface avec l'application, ou l'utilisateur
- modules du logiciel proprement dit
- interface avec la matériel

Nous allons étudier rapidement les fonctions de ces modules, et dans le cas du deuxième groupe les situer par rapport aux unités de contrôle, description-visualisation et communication.

2.2.1. Interface système/application

* Le module "PRELUDE" a pour fonction de proposer à un programme d'application l'ensemble des primitives disponibles ainsi que les types d'éléments permis par le système, sous forme de procèdures déclarées en "EXTERNAL" pour les premières, de constantes pour les seconds. Ce module recoupe tout à fait la présentation faite en paragraphe 2 du chapitre IV.

* Le module "DIALOGUE" propose lui à l'utilisateur, via un menu interactif, toutes les primitives de "PRELUDE". Ce module nous a servi en outre à tester correctement le logiciel.

2.2.2. Modules système

Ces modules, au nombre de cinq, réalisent les opérations des différentes unités. Il est à noter que, pour éviter un trop grand nombre de déclarations de procédures ou variables en "EXTERNAL", un module regroupe des fonctions de différentes unités. Un découpage "logique" a toutefois été réalisé afin de permettre une meilleure lisibilité et extensibilité.

* Le module "CLOVIS" est le module principal du logiciel, et constitue le programme éxécutable. Il propose les fonctions suivantes :

- gestion de la structure de données, en ce qui concerne la gestion des pointeurs et celle des enregistrements de type "noeud" -cf. paragraphe 3.1.1. chapitre IV-: partie "communication"

- primitives utilisateurs, déclarées en "GLOBAL", offertes à l'utilisateur -interactif ou application-: partie "controle"

* Le module "COMUNIV" gère l'ordonnancement des processus, selon les critères essentiels de type d'objet. Il opère donc l'aiguillage sur les primitives de visualisation et description adéquates. En ce qui concerne l'attribution, le module "CLOVIS" assure cet ordonnancement de par la constitution des primitives offertes à l'utilisateur -cf.paragraphe 2.2.2. chapitre IV-.

* Le module "DESATTS" a un rôle qui s'étend sur les trois unités:

- gestion interne des différents processus de visualisation par type d'attribut morphologique: partie "controle"

- gestion de la structure de données pour ce qui est des enregistrements de type "attribut" et libération des pointeurs de noeuds: partie "communication"

- offre les opérateurs de visualisation et description: transformations géométriques pures, d'affichage, de prise de vue, calcul d'interpolation pour l'identification et d'approximations en segments pour la visualisation, ... : partie "description-visualisation".

* Le module "GEOMETR" offre simplement des opérateurs de calcul

matriciel utiles pour les transformations géométriques et entre donc dans l'unité de "description-visualisation".

* Le module "HELPILOT", dernier module du système, assure la liaison avec le synthétiseur aval, en l'occurence le matériel HELIOS, par l'invocation des primitives de base du matériel. Ce module fait donc à ce titre partie de l'unité de contrôle.

La figure V.14 schématise les relations entre modules et unités.

UNITES / MODULES	CONTROLE	COMMUNICATION	DESCRIPTION /VISUALISATION
CLOVIS	X	X	
COMUNIV	X		
DESATTS	X	X	X
GEOMETR			X
HELPILOT	X		

Figure V.14: Relations modules-unités

2.2.3. Interface système/matériel

Deux modules, l'un implanté sur le calculateur principal VAX -"GRAFHELIO"-, l'autre sur le calculateur satellite MICROENGINE -"COMVAX"- constituent l'interface de gestion d'échanges que nous avions évoqué lors de la présentation de la configuration matérielle de réalisation (cf. paragraphe 1.1. chapitre IV). Ces fonctions seront tout spécialement développées dans le paragraphe suivant. Succintement, le module "GRAFHELIO" propose la liste des primitives d' émission et réception vers le matériel, le module "COMVAX" jouant un rôle symétrique vis-à-vis du système.

2.2.4. Structuration générale

La figure V.15 schématise les relations entre modules et leur appartenance aux trois unités ainsi que leur place dans le système par rapport à l' application et au matériel.

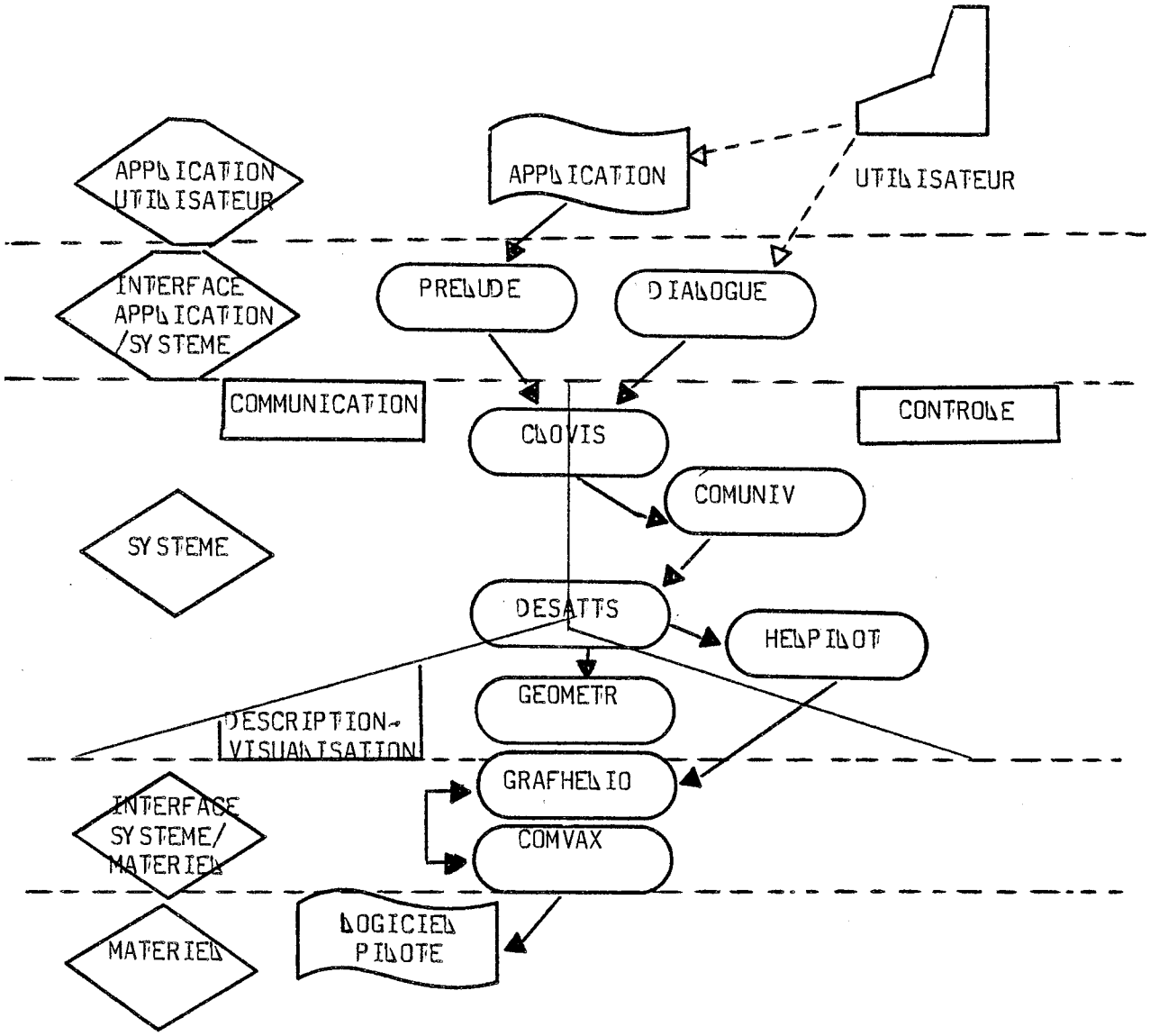


Figure V.15: Structure générale du logiciel

Remarque: les sources de ces différents modules représentent environ 8000 lignes PASCAL, se répartissant comme suit:

	PRELUDE	: 100	
	DIALOGUE	: 1100	
	CLOVIS	: 2700	
	COMUNIV	: 600	
VAX	DESATTS	: 2000	Noyau = 6000 lignes
	GEOMETR	: 100	
	HELPILOT	: 500	
	GRAFHELIO	: 250	
MICROENGINE	COMVAX	: 600	

3. Réalisations dans le cadre de l'unité de contrôle

Les deux principales tâches réalisées par l'unité de contrôle, outre la mise à disposition de l'utilisateur des primitives et types d'éléments du système, point déjà développé dans ce mémoire, sont la gestion des processus, en particulier de visualisation, et la liaison avec le terminal HELIOS.

3.1. Gestion des processus

Nous traiterons ici essentiellement le cas des processus de visualisation , les processus d'attribution ayant été déjà envisagés dans l'énoncé des primitives destinées à l'utilisateur. Quant aux processus de consultation et de description, ils sont tout à fait symétriques des deux processus précédents.

Cette gestion se fait dans deux modules principalement, "COMUNIV" et "DESATTS" avec des rôles que nous étudierons successivement, et nécessite l'initialisation de structures de données dont nous avions déjà donné les grands principes -cf. paragraphe 3.2.4. chapitre IV-.

3.1.1. Fonctions des modules COMUNIV et DESATTS

* Rôle de "COMUNIV": il oriente la primitive "VISUALISER" sur les primitives de visualisation adéquates, selon le type morphologique. La procédure "VISION" se présente donc sous la forme d'un aiguillage comme suit, précédé de l'appel de la primitive permettant la décomposition des objets en faces HELIOS. Nous présentons cette procédure, dont le paramètre principal est "TYPES", type morphologique de l'objet, sous une forme très proche de PASCAL. Cette remarque est valide pour les autres descriptions de procédures qui pourront suivre dans ce chapitre.

```
Procedure VISION (TYPES);
begin
   DECOMPOSITION (TYPES);
   case TYPES of
      MPOINTS2D: VISIONPOINTS2D;
      MSEGMENTS2D: VISIONSEGMENTS2D;
      MLIGNES2D: VISIONLIGNES2D;
      MCERCLE2D: VISIONCERCLE2D;
      MDISQUE2D: VISIONDISQUE2D;
      MT2D: VISIONT2D;
      MFACE2D: VISIONFACE2D;
      MQUADRI2D: VISIONQUADRI2D;
      OTHERWISE: (* ... types non implantes *)
   end;
end;
```

* Rôle de "DESATTS": il décrit chacune des procédures "VISION..." pour en présenter l'agencement interne. Chacune de ces primitives invoque des procédures de composition géométriques, de clipping et passage du repère utilisateur au repère écran -éléments que nous verrons dans le cadre de l'unité de description-visualisation-, et une procédure de tracé appelant les actions matériel, entrant dans le cadre du module "HELIOPLOT". Ces actions seront décrites dans le paragraphe 3.2 traitant des échanges avec HELIOS.

Voici la description des différentes procédures de visualisation du module "DESATTS".

Procedure VISIONPOINTS2D;

```
begin
   CALCUL (matrice-composition);
   while POINTS (X,Y) do
   begin
      CALCUL-POINTS-COMPOSES (X1,Y1);
      CLIPPING2D (X2,Y2);
      FENETRACLOTURE (X3,Y3);
      TRACERPOINTS2D (round(X3), round(Y3), F1(ASPECT));
   end;
end;
```

Remarques:

- Xi et Yi sont des réels

- La fonction F1 permet de coder les informations d'aspect pour le matériel et, si celles-ci ne sont pas renseignées, de transmettre des données standard

- "CALCUL", "CALCUL-POINTS-COMPOSES", "CLIPPING2D" et "FENETREACLOTURE" seront étudiées dans le paragraphe 5 de ce même chapitre

- "TRACERPOINTS2D" est l'action matériel correspondante à la visualisation de "MPOINTS2D", intégrée dans le module "HELIOPLOT".

```
            Procedure VISIONSEGMENTS2D;
        begin
           CALCUL (matrice-composition);
           while SEGMENTS ((XD,YD),(XF,YF)) do
           begin
              CALCUL-POINTS-COMPOSES (XD1,YD1);
              CALCUL-POINTS-COMPOSES (XF1,YF1);
              CLIPPING2D (XD2,YD2,XF2,YF2);
              FENETREACLOTURE (XD3,YD3);
              FENETREACLOTURE (XF3,YF3);
              TRACERSEGMENTS2D (round(XD3), round(YD3), round(XF3),
                                round(YF3), F1(ASPECT));
           end;
        end;

    Procedure VISIONLIGNES2D;
        begin
           CALCUL (matrice-composition);
           while SOMMETS (X,Y) do
           begin
              CALCUL-POINTS-COMPOSES (X1,Y1);
              CLIPPING2D (X2,Y2);
              FENETREACLOTURE (X3,Y3);
              TABX <-- round (X3); TABY <-- round (Y3);
           end;
           TRACERLIGNES2D (TABX, TABY, NBSOMMETS, F1(ASPECT));
        end;

        Procedure VISIONCERCLE2D;
        begin
            CALCUL (matrice-composition);
            CALCUL-POINTS-COMPOSES (XCIR,YCIR);
            CALCUL-POINTS-COMPOSES (XCEN,YCEN);
            RAYON <-- SQRT ((XCIR-XCEN)**2 + (YCIR-YCEN)**2);
            (* APPROXIMATION CAVALIERE DU CERCLE *)
           DETERMINATION (NBPAS);
           DETERMINATION-POINT (XD,YS);
           for 1 to NBPAS do
           begin
              DETERMINATION-POINT (XF,YF);
              CLIPPING2D (XD1,YD1);
              CLIPPING2D (XF1,YF1);
              FENETREACLOTURE (XD2,YD2);
              FENETREACLOTURE (XF2,YF2);
              TABX <-- round (XD2), round (XF2);
              TABY <-- round (YD2), round (YF2);
              YD <-- YF; XD <-- XF;
           end;
           TRACERLIGNES2D (TABX, TABY, NBSOMMETS, F1(ASPECT));
         end;
```

Remarque: l'approximation cavalière du cercle n'est pas utile si le matériel est capable de générer un cercle directement. Dans ce cas, un

autre processus de visualisation de cercle est proposé, appelant directement une primitive matériel de tracé de cercle, incluant alors le clipping et la gestion de la cloture.

Procedure VISIONDISQUE2D;

```
begin
   CALCUL (matrice-composition);
   CALCUL-POINTS-COMPOSES (XCIR,YCIR);
   CALCUL-POINTS-COMPOSES (XCEN,YCEN);
   RAYON <-- SQRT ((XCIR-XCEN)**2 + (YCIR-YCEN)*2);
   (* APPROXIMATION CAVALIERE DU DISQUE *)
   (* DETERMINATION DE PORTIONS DE FACES EXTREMES
            DE LA PERIPHERIE VERS LE CENTRE  *)
   repeat
      DETERMINATION-FACE1 (XD,YD,XF,YF);
      CLIPPING2D (XD1,YD1,XF1,YF1);
      FENETREACLOTURE (XD2,YD2,XF2,YF2);
      TABX <-- round (XD2), round (XF2);
      TABY <-- round (YD2), round (YF2);
      DETERMINATION-FACE2 (XD,YD,XF,YF);
      CLIPPING2D (XD1,YD1,XF1,YF1);
      FENETRACLOTURE (XD2,YD2,XF2,YF2);
      TABX <-- round (XD2), round (XF2);
      TABY <-- round (YD2), round (YF2);
   until FIN-APPROXIMATION;
   TRACERFACE2D (TABX, TABY, NBSOMMETS, F1(ASPECT),
                 F2(ECLAIRAGE));
end;
```

Remarques:

- pour ce qui est de l'approximation, même remarque que pour le cercle
- la fonction F2 a un rôle similaire à F1, et traite les informations d'éclairage s'appliquant à un objet de sous-classe "SURF".

Procedure VISIONT2D;

```
begin
   CALCUL (matrice-composition);
   CALCUL-POINTS-COMPOSES (XDEB,YDEB);
   FENETREACLOTURE (XDEB1,YDEB1);
   CLIPTEXTE (TEXTE);
   TRACERT2D (TEXTE, NBCAR, round(XDEB1), round (YDEB1),
              F1(ASPECT));
end;
```

```
Procedure VISIONFACE2D;

begin
   CALCUL (matrice-composition);
   while SOMMETS (X,Y) do
   begin
      CALCUL-POINTS-COMPOSES (X1,Y1);
      CLIPPING2D (X2,Y2);
      FENETREACLOTURE (X3,Y3);
      TABX <-- round (X3);
      TABY <-- round (Y3);
   end;
   TRACERFACE2D (TABX, TABY, NBSOMMETS, F1(ASPECT),
                 F2(ECLAIRAGE));
end;
```

Remarque: la procédure "TRACERT2D" sera développée dans le paragraphe 3.2. à titre d'exemple d'action matériel.

```
Procedure VISIONQUADRI2D;
begin
   CALCUL (matrice-composition);
   CALCUL-POINTS-COMPOSES (XIG,YIG);
   CALCUL-POINTS-COMPOSES (XSD,YSD);
   CLIPPING2D (XIG1,YIG1);
   CLIPPING2D (XSD1,YSD1);
   FENETREACLOTURE (XIG2,YIG2);
   FENETREACLOTURE (XSD2,YSD2);
   TRACERQUADRI2D (round (XIG2), round(YIG2), round (XSD2),
                   round (YSD2), F1(ASPECT), F2(ECLAIRAGE));
end;
```

Remarques générales:

* L'algorithme de clipping travaillant sur des segments, la procédure "CLIPPING2D" inclut une décomposition de l'objet en segments
* Les procédures d'identification -processus de description- ont, nous le comprendrons aisément, une structure tout à fait similaire à celles de visualisation, avec l'éclatement au niveau de "COMUNIV" d' une procédure "TESTCOORDONNEES" en "POINTVECTEUR2D", "POINTCERCLE2D", "POINTDISQUE2D", "POINT2D", "POINTFACE2D", "POINTQUADRI2D" implantées dans "DESATTS". Seul apparait un regroupement pour les types morphologiques "MPOINTS2D", "MLIGNES2D" et "MSEGMENTS2D" en un ensemble "VECTEUR2D".

3.1.2. Initialisations des structures de données associées

Une structure de données complète a été mise en place afin de répondre aux différents problèmes soulevés par l'agencement des processus et l'éxécution des procédures. Elle essaie d'autre part de répondre à un effort de cohésion entre les versions pour différents matériels. Elle est initialisée dans le module "CLOVIS", par la primitive "CHOIXTERMINAL". La figure V.16 représente la structure et indique les phases où elle dicte la décision à prendre. Puis une description proche de PASCAL sera donnée ainsi que quelques explicitations.

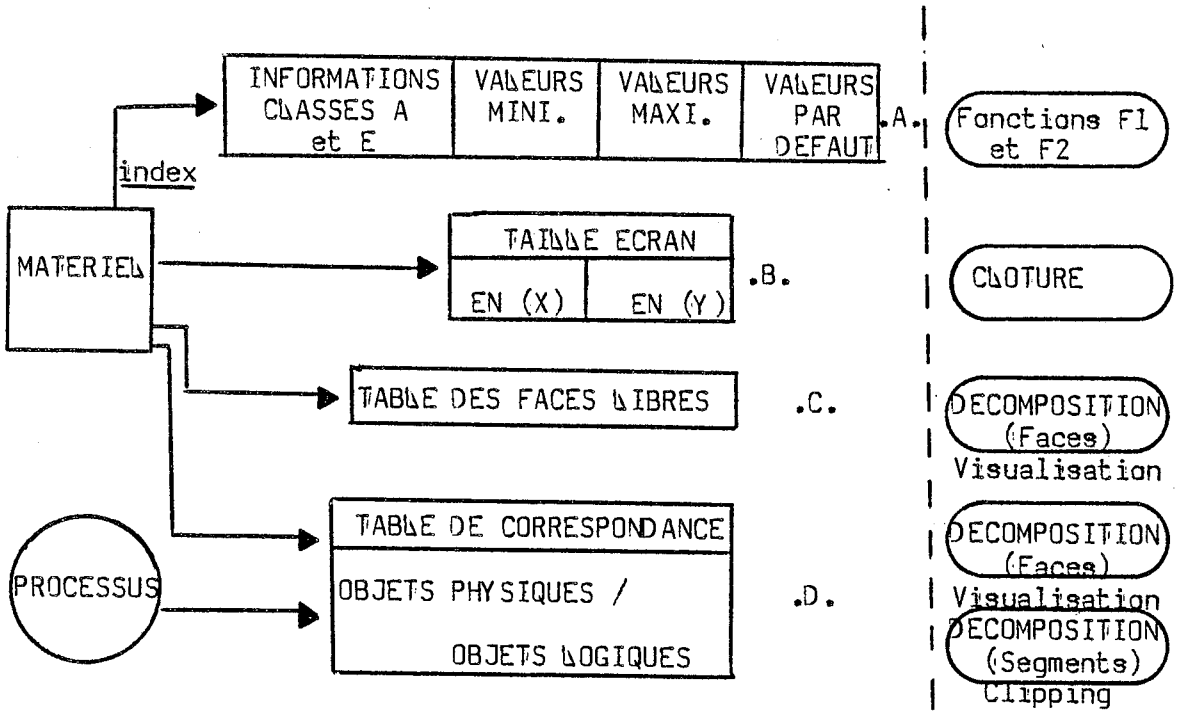


Figure V.16: Structure de données associées à l'ordonnancement des processus

* Description:

```
TABLE = array (0..NBMATERIEL) of ELEMENT;

ELEMENT = record
    DEB,FIN,STANDARD : SETINFOS; (*cf. .A. *)
    ECRAN : array (1..2) of integer; (*cf. .B. *)
    TABLIBRE : array (1..NBOBJPHYS,0..MAXLIBRE)
                    of boolean ; (*cf. .C. *)
    TABPROC : array (0..NBPROCESSUS) of
                    ELTPROCESSUS ; (*cf. .D. *)
end;
```

```
      SETINFOS = record
    COULEUR : integer;
    VISIBILITE : integer;
    GRAPHIQUE : integer;
    REMPLI : integer;
    ORIENTATION : integer;
    TAILLE : integer;
    POLICE : integer;
    ECRITURE : integer;
    REFLEXION : integer;
    LUMAMB : integer;
    COULRVB : integer;
    ANGA : real;
    ANGB : real;
end;

ELTPROCESSUS = array (0..NBOBJLOG,1..NBOBJPHYS)  of boolean;
```

Les constantes :

- NBMATERIEL : nombre de matériels = 10
- NBOBJPHYS : nombre d'éléments de base matériel = 10
- MAXLIBRE : nombre d'éléments libres par type d'élément de base = 1023
- NBPROCESSUS : nombre de processus = 10
- NBOBJLOG :nombre d'objets logiques = 20

* Initialisations :

- Indice MATERIEL :

  0 : HELIOS

- Indices PROCESSUS :

  0 : VISUALISATION
  1 : CLIPPING2D

- Indices OBJETS PHYSIQUES :

  1 : FACE
  2 : SEGMENT
  3 : POINT

- Indices OBJETS LOGIQUES :

  0 : MPOINT2D
  1 : MSEGMENT2D
  2 : MLIGNE2D
  3 : MCERCLE2D
  4 : MDISQUE2D
  5 : MTEXTE2D
  6 : MFACE2D
  7 : MQUADRI2D

- Structure :

PROCESSUS	OBJ. PHYS	ELTS LIBRES	MPOINT	MSEGMENT	MLIGNE	MCERCLE	MDISQUE	MTEXTE	MFACE	MQUADRI
			OBJETS LOGIQUES							
0	FACE	1024	1	1	1	1	1	1	1	1
	SEG.	-	0	0	0	0	0	0	0	0
	POINT	-	0	0	0	0	0	0	0	0
1	FACE	-	0	0	0	0	0	0	0	0
	SEG.	-	1	1	1	1	1	0	1	1
	POINT	-	0	0	0	0	0	0	0	0

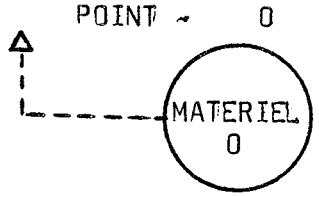


Figure V.17: Initialisation de la structure TABLE

* Explicitations : reprenons par exemple le cas de la visualisation d'un type d'objet "MFACE2D".

1 - "VISION" : appel de "DECOMPOSITION" :
a/- vérifier si TABLE(0).TABPROC(0).ELTPROCESSUS(6,1) est vrai.
b/- calculer le nombre de faces correspondant à "MFACE2D" -ici 1 face-.
c/- rechercher 1 face libre dans TABLE(0).TABLIBRE(1,i)
d/- créer le descripteur d'attribut "MATERIEL" avec la face no. i pour l'objet de type "MFACE2D"
e/- transmettre au matériel le no. face i

2 - "VISIONFACE2D" :
a/- appel de "CLIPPING2D" :
1- vérifier si TABLE(0).TABPROC(1).ELTPROCESSUS(6,2) est vrai
2- calculer les coordonnées des segments
3- calculer le clipping
b/- appel des fonctions "F1(ASPECT)" et "F2(ECLAIRAGE)": vérifier si les informations mémorisées dans les attributs des classes A et E correspondent à l'intervalle , dont les bornes sont TABLE(0).DEB et TABLE(0).FIN. Si les informations ne sont pas renseignées, transmettre au matériel les informations induites par TABLE(0).STANDARD.

3.2. Liaison avec HELIOS

Cette interface est assurée par les trois modules "HELPILOT", "GRAFHELIO" et "COMVAX". En ce qui concerne ce dernier, nous pouvons dire qu'il assume le rôle d'interface de simulation, que nous avions évoqué dans le paragraphe 1.1. du chapitre IV.

La liaison avec HELIOS comporte deux phases :

- invocation des actions matériel par le module 'HELPILOT"

- gestion des échanges entre les deux calculateurs, VAX d'une part, MICROENGINE d'autre part, via les modules "GRAFHELIO" et "COMVAX".

3.2.1. Logiciel pilote d'HELIOS

Ce logiciel est invoqué sous forme de commandes dans les versions les plus récentes d'HELIOS. Nous avons donc conservé cet aspect, le module "HELPILOT" respectant la syntaxe des commandes.

De facon générale, une commande vers le terminal HELIOS se présente sous la forme:

< ESC> <CODE COMMANDE> <paramètres> ...

La figure V.18 récapitule les commandes permise par le terminal. En retour le terminal émet un acquittement de la forme:

< ACK> <paramètres> ou <CODE RETOUR> <CR>

CODE COMMANDE	PARAMETRES	ACTION DECLENCHEE	DOMAINE D'APPLICATION		
			INTERVALLE DE FACES	TOUTES FACES	AUCUNE FACE
I	-	Initialisation			x
1	no.face	Face courante	x		
2	faces deb,fin	Faces courantes	x		
P	no.plan	Plan courant	x		
W	texture trait, texture remplissage	Aspect graphique	x		
K	orientation,taille écriture,police	Aspect alpha-numérique	x x		
T	pointeur de texture	Texture ou couleur	x		
R	no.modèle	Réflexion	x		
V	code visibilite	Visibilite	x		
N	angles a,b	Normale à la face	x		
U	coordonnées(x,y) des vecteurs (u,v)	Repère de la face	x		
S	couleur source lumière ambiante	Couleur éclairage		x	
D	angles a,b	Direction source		x	
.	(x,y)	Point	x		
(	(xd,yd) (xf,yf)	Segment	x		
=	(xig,yig) (xsd,ysd)	Rectangle	x		
/	nbre points,(xi,yi)	Polygone	x		
;	centre(x,y),rayon	Cercle	x		
"	(xdeb,ydeb),nbre	Texte	x		

	caractères,texte				
£	"	Effacement plan	x		
F	(x,y)	Origine plan		x	
O	(x,y)	Origine affichage		x	
Z	(zx,zy)	Zoom		x	
C	code visibilité, repére associé, couleur	Définition du réticule			x
+	nbre de coordonnées coordonnées	Collecte de			x
A	centre,coefficient d'échelle	Affichage palette			x
B	couleur	Description couleur			x
a	"	Identification couleur			x

Figure V.18: Tableau des commandes HELIOS

Ainsi, le module "HELPILOT" propose une suite de procédures d'invocation des commandes d'HELIOS :

- initialisation : "I","P"

- déclaration des faces courantes, déclenchée par "DECOMPOSITION", correspondant aux commandes "1" et "2"

- tracé d'objets de type "FILDEFER" : primitives "TRACERPOINTS2D", "TRACERSEGMENTS2D", "TRACERLIGNES2D", "TRACERT2D" (cf. paragraphe 3.1.1. de ce chapitre) comprenant l'analyse des informations d'aspect "ACAR" ou "ATRAIT", soit les commandes "W", "K", "T", "V" et le tracé lui-même : '.', '(', '"'

- tracé d'objets de type "SURF" : primitives "TRACERFACE2D", "TRACER QUADRI2D" analysant les informations d'aspect de type "ASURF", soit "W", "T", "R", "V", d'éclairage ("N", "U", "S", "D") et de tracé ("=", "/")

- description de couleur : "A", "B", "a"

Pour émettre ces commandes, "HELPILOT" utilise différentes procédures d'émission selon le type de la donnée transmise (caractère, entier, réel, ...), procédures constituant le module GRAFHELIO. Ce sont ces problèmes d'échanges que nous abordons maintenant.

3.2.2. Echanges VAX-HELIOS

Deux problèmes sont apparus dans cette gestion des échanges entre VAX et HELIOS : émission des informations sous forme de caratères ASCII (avec décodage et encodage), et la synchronisation des deux procédures d'émission et réception de part et d'autre.

Avant d'aborder ces deux points, énonçons rapidement quelques éléments de cette communication :

- le module "COMVAX", implanté sur le microcalculateur d'HELIOS, gère les échanges selon deux modes:
  . transparent: gestion simplement entre la console et le VAX, ignorant HELIOS
  . opération: interprétation et éxécution des commandes vers HELIOS, par reconnaissance du caractère <ESC>

- les deux lignes que gère "COMVAX" sont à 9600 bauds, d'où une réalisation un peu délicate devant le risque de perte de caractères dû à la conjugaison de la vitesse "rapide" de transmission et les performances du microcalculateur

- "comvax" gère l'écho sur la console, celui-ci étant inhibé sur le VAX par une commande de configuration du terminal.

La figure V.19 donne un schéma de la connexion VAX-HELIOS.

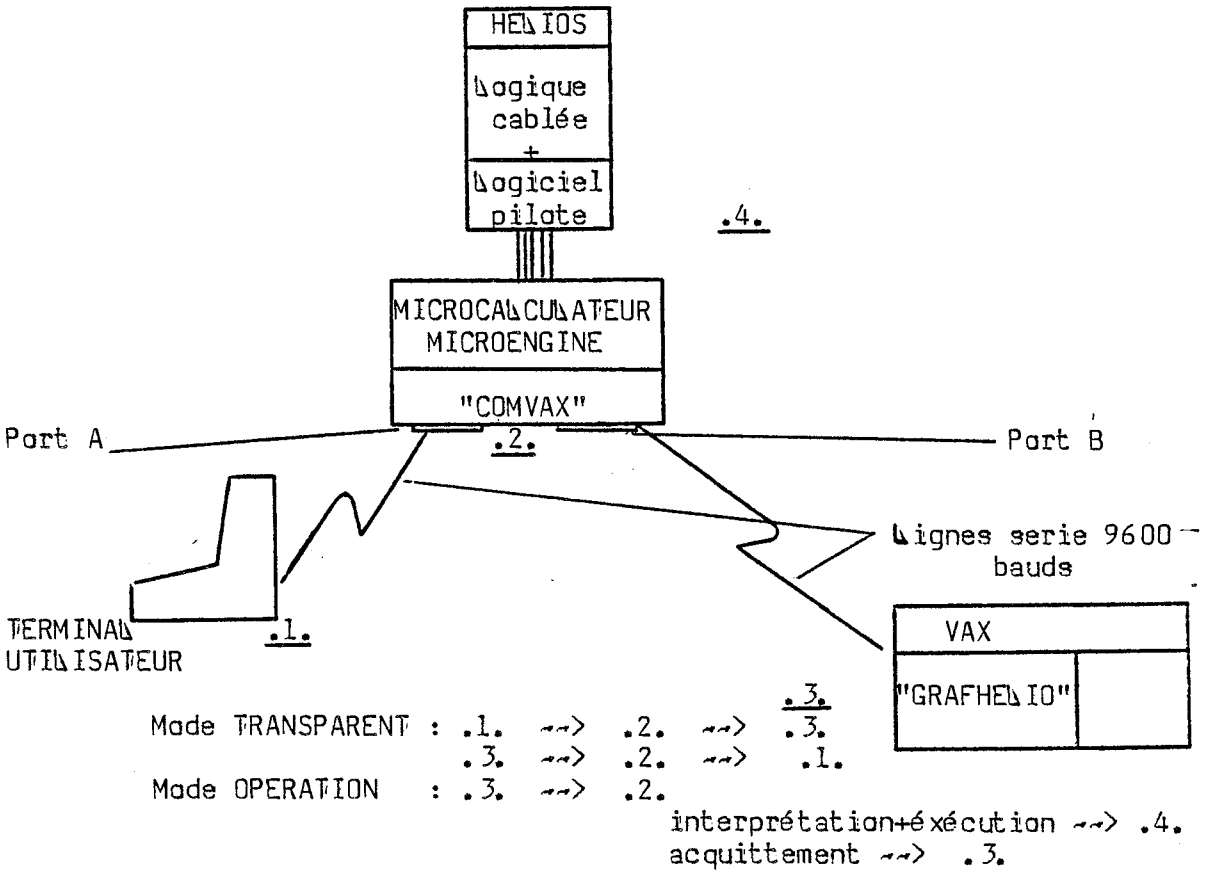
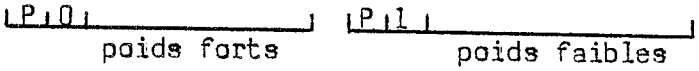


Figure V.19: Modes de communication VAX-HELIOS

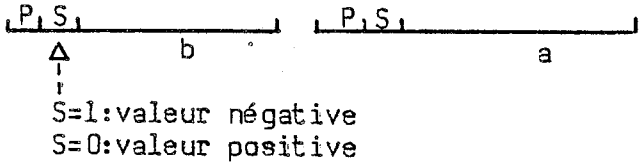
* Codage des informations : HELIOS manipule des paramètres de différents types, et donc d'un codage spécifique:

- entier court ou caractère : valeur comprise entre 0 et 127

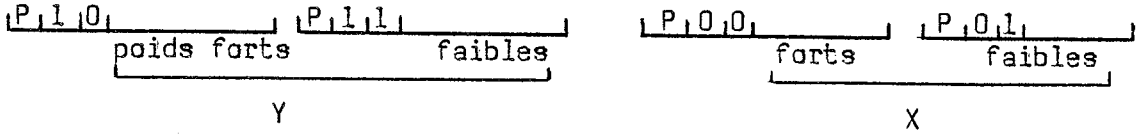
- entier long : valeur entre 0 et 4095



- direction : deux angles a et b exprimés en 1/32e. de pi., a varie entre -pi et +pi, b entre -pi/2 et +pi/2. Les valeurs négatives sont codées en complément à 2.



- coordonnées : (x,y) peuvent varier entre 0 et 1023. Certaines commandes nécessitant des coordonnées négatives, celles-ci sont exprimées modulo 1024.



Ainsi les modules "GRAFHELIO" et "COMVAX" proposent des primitives d'émission et de réception encodant et décodant ces différents types de paramètres : SEND (octet), SENDLONG (entier), SENDCOOR (x,y), SENDDIR (a,b), SENDC (string), RECEIVE (car), RECEIVELONG (entier), RECEIVECOOR (x,y), RECEIVEDIR (a,b), RECEIVEC (string)

* Synchronisation émission/réception et traitement des commandes HELIOS sont réalisés par "COMVAX". La figure V.20 schématise ces fonctions

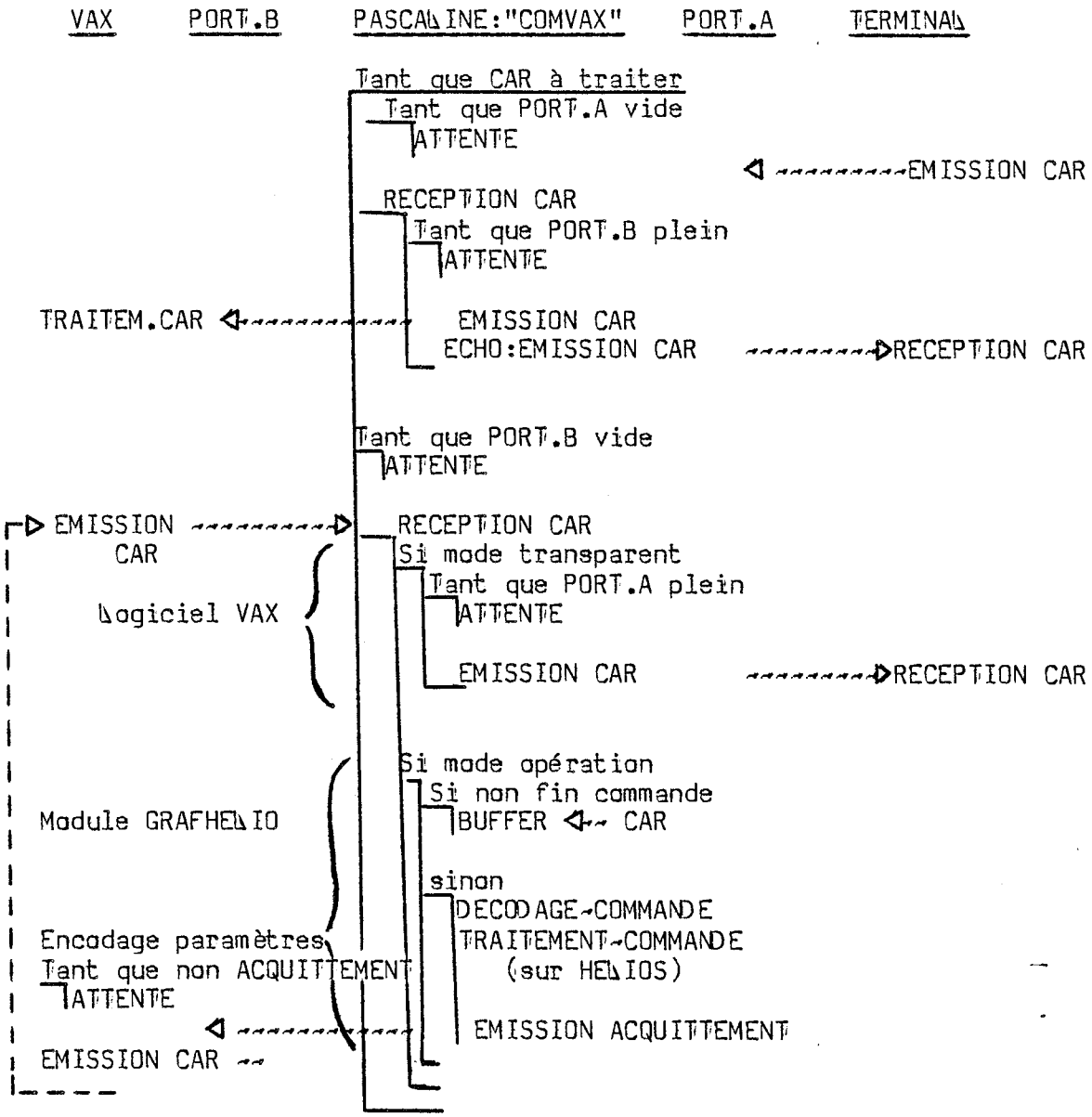


Figure V.20: Schèma des fonctions de COMVAX

Le traitement des commandes pour "HELIOS" s'opère d'une facon particulière: interpréter ces commandes en termes de primitives du logiciel pilote, orientées par le microcalculateur vers la logique cablée du matériel. Illustrons ceci par la commande de tracé de texte.

```
Module "HELPILOT":
     Procédure TRACERT2D;
     begin
        ANALYSEASPECT ( ASPECT);
        SEND (ESC);
        SEND (NBCAR-COMMANDE) ; (*nombre de caractères de la commande
                                   pour interprétation par COMVAX*)
        SENDC ('"') ; (*code commande*)
        SENDCOOR (X,Y);
        SEND (LGCHAINE);
        SENDC (TEXTE);
        SENDC ('F'); (*fin commande*)
     end;

Module "GRAFHELIO":déclaration des procédures SEND, SENDC,
                   SENDCOOR


Module "COMVAX":
     Procédure RECEPTION (CAR);
     begin
        DETECTION-ESC;
        while (CAR<>"F") do
        begin
           READ (CAR);
           BUFFER <-- CAR;
        end;
        TRAITEMENT (BUFFER);
     end;

     Procédure TRAITEMENT (BUFFER);
     begin
        CDE <-- ord (BUFFER);
        case BUFFER (1) of (*code commande*)
        '"':(*TEXTE*)
            begin
               XDEB <-- DECODAGE (CDE(4), CDE(5));
               YDEB <-- DECODAGE (CDE(2), CDE(3));
               LGCHAINE <-- DECODAGE (CDE(6));
               MESSAGE <-- DECODAGE (CDE(7), ..., CDE(7+LGCHAINE-1);
               TEXTE (XDEB, YDEB,LGCHAINE, MESSAGE); (*appel
                                          primitive logiciel pilote*)
            end;
         end;
      end;
```

4. Réalisations dans le cadre de l'unité de communication

Ces réalisations concernent essentiellement la gestion du descripteur "MATERIEL" et plus précisément dont le type d'attribut est "MTFACE" -cf. paragraphe 2.1.1. de ce même chapitre-.

L'intérêt d'un tel descripteur est de mémoriser dans la structure de données graphiques le lien entre le type d'élément proposé à l'utilisateur et l'intervalle de faces correspondant sur le matériel HELIOS, ces faces étant, rappelons-le, les identifieurs pour le matériel d'un certain nombre d'informations -en particulier d'aspect- et les éléments de base exploités par le matériel pour la visualisation -cf.paragraphe 2.3. chapitre III-.

La gestion du descripteur "MATERIEL" permet d'envisager une seconde gamme de processus -cf.paragraphe 3.2. chapitre IV-, tant de visualisation -les numéros de faces sont communiqués au matériel à chaque visualisation sans besoin de décomposition systématique et de recherche de nouveaux numéros de faces- que de description -l'identification se fait alors simplement par l'identification par le matériel des faces et l'extraction de l'objet par le lien objet-faces de la structure-. En ce qui concerne les processus d'attribution et de consultation, ils peuvent évoluer dans la mesure où l'on peut choisir de mémoriser les informations -certaines du moins- au niveau matériel.

Nous examinerons donc dans un premier temps la gestion de ce descripteur un peu particulier avec ses problèmes, règles et dans un second temps l' utilisation que l'on peut en faire dans les différents processus d'un système de synthèse d'images.

4.1. Gestion du descripteur matériel

4.1.1. Quelques précisions sur les faces d'HELIOS

* Les faces sont identifiées par un entier long, compris entre 0 et 1023
* Les faces sont les identifieurs d'une table contenant les informations suivantes:

- pointeur de texture ou couleur
- brillance (modèle de réflexion)
- normale à la face
- matrice de de projection des textures
  * Est tenu à jour l'indicateur de visibilité des faces pour chacun des 8 plans d'identification
  * Une même face peut appartenir à plusieurs objets situés en des points différents de l'écran, si elle exprime les mêmes informations de texture, brillance, normale (et repère) et visibilité.

4.1.2. Rappel du descripteur

Il s'agit d'un descripteur de la pseudo-classe "MATERIEL", de type "MTFACE". Sa description est donnée par la figure V.21.

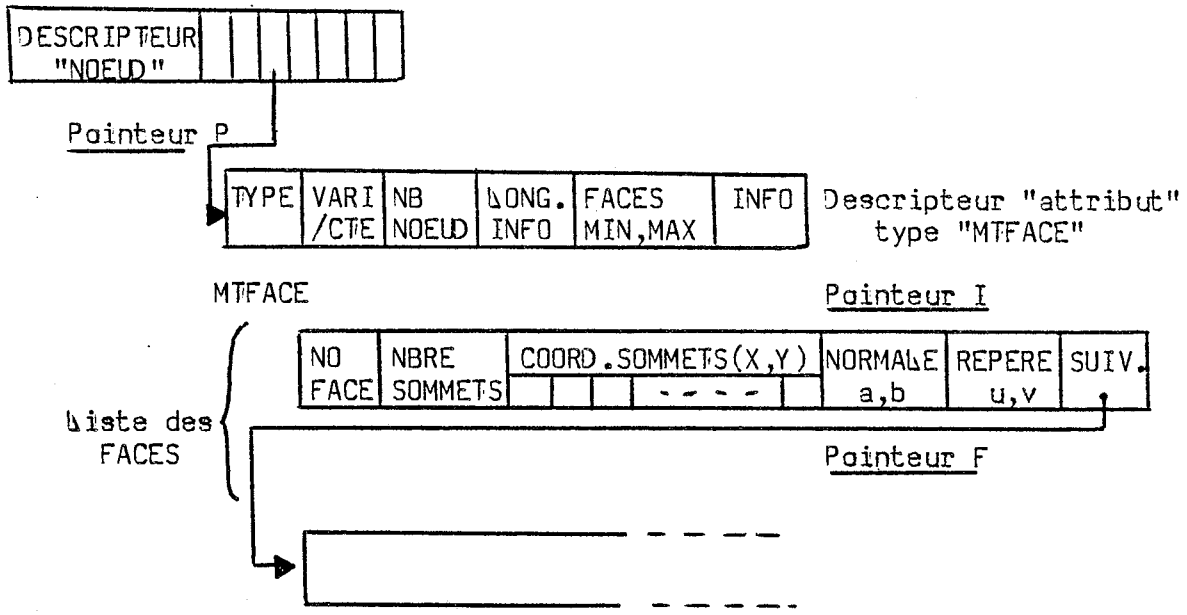


Figure V.21: Descripteur MATERIEL, type MTFACE

4.1.3. Règles de gestion

Nous examinerons ces règles dans le cas des primitives suivantes:

- VISUALISER : conduit à la création ou non de ce descripteur
- DETRUIRE : destruction du descripteur de type "noeud"
- AFFECTER : informations des classes "MATERIEL", "ASPECT" et "MORPHOLOGIE"

Cette gestion concerne donc les modules "CLOVIS" et "DESATTS". Les règles sont données sous forme algorithmique, les variables P, F et I étant les pointeurs présents sur la figure V.21.

```
Primitive VISION (TYPEMORPHO);
debut
  si P = nil
    debut
      DECOMPOSITION (TYPEMORPHO, NBFACES, COORD-SOMMETS);
      N <-- NBFACES;
      si N <> 0
        debut
        (*Recherche des N nos.faces libres Ni*)
        pour i = 1 à N
          debut
          si ∃ face (f) tq texture(f)=texture(Ni)
                           mod.reflex(f)=mod.reflex(Ni)
                           visibilite(f)=visibilite(Ni)
                           normale(f)=normale(Ni)
                           repere(f)=repere(Ni)
            debut
              Ni <-- f;
              NBOBJETS (f) <-- (+1);
            fin
          sinon
            debut
              RECHERCHE (Ni) dans TABLIBRE;
            fin
          fin
        CREATION-DESCRIPTEUR;
        GESTION-POINTEURS (P, I, F);
        fin
      fin
      sinon
        debut
          CONSULTATION (N.nos faces);
          EMISSION (N) --> HELIOS;
          VISUALISATION (TYPEMORPHO);
        fin
    fin.
```

La structure "TABLIBRE", vue dans le paragraphe 3.1.2. de ce chapitre, est complétée par l'information NBOBJETS, donnant pour chaque face HELIOS le nombre d'objets liés à celle-ci.

```
Primitive AFFECTER (CLASSE:MATERIEL);
debut
  si P <> nil
    debut
      IDENTIFIER-FACE (NX);
      RENSEIGNER-INFOS (NORMALE, REPERE);
      NX <-- NX+INFOS;
      (*Recherche face (f) identique à (NX)*)
      si ∃ (f) tq (f) = (NX)
        debut
          NBOBJETS(f) <-- (+1);
          GESTION (F(f), I(f));
          (*Libérer (NX) pour P courant*)
          LIBERER (F(NX), I(NX));
          MODIF (FACEMIN,FACEMAX);
          NBOBJETS (NX) <-- (-1);
          si NBOBJETS (NX) <= 0 alors LIBERER-TABLIBRE (NX);
        fin
      (*Recherche d'une nouvelle face (NY) et modif. (NX)
                  pour autres objets de pointeur Pi*)
        sinon
          debut
            si NBOBJETS (NX) > 1
              debut
                RECHERCHE (NY);
                CREER (F(NY));
                CREER (I(NY));
                MODIF-Pi (FACEMIN,FACEMAX);
              fin
            sinon MODIF-INOS (F(NX));
          fin
      fin
    fin.
```

LIBERER (NX)

Pour expliciter cet algorithme ainsi que le suivant, tous deux à la fois tortueux et trop simplistes, il faut noter que la difficulté réside dans l'appartenance possible d'une face à plusieurs objets, ce qui entraine, en cas de modifications touchant les critères de base d'une face de :

- pour l'objet courant, rechercher une nouvelle face répondant aux nouveaux critères

- si cette recherche est fructueuse, libérer la face pour l'objet courant et rattacher la nouvelle

- sinon, dans le cas où la face modifiée appartient à plusieurs objets, faire cette même démarche pour ces autres objets.

```
Primitive AFFECTER (CLASSE : ASPECT);
debut
  si P <> nil
    debut
      si INFO-ASPECT = COULEUR ou TEXTURE ou VISIBILITE
                                  ou REFLEXION
        debut
          PARCOURS (N nos faces Ni);
          pour i = 1 à N
          debut
            si ∃ (f) tq (f) = (Ni)
              debut
              NBOBJETS (f) <-- (+1);
              GESTION (F(f), I(f));
              LIBERER (Ni);
            fin
          sinon
            debut
              si NBOBJETS (Ni) > 1
                debut
                  RECHERCHE (Nj);
                  CREER (F(Nj), I(Nj));
                  MODIF-Pj (FACEMIN, FACEMAX);
                  LIBERER (Ni);
                fin
              fin
            fin
          fin
        fin
      fin.
```

```
Primitive AFFECTER (CLASSE:MORPHOLOGIQUE);
debut
  si P <> nil
  debut
    PARCOURS (N faces Ni);
    pour i = 1 à N alors LIBERER (Ni);
    DETRUIRE (I, F);
  fin
fin.
```

C'est le cas le plus simple : destruction des pointeurs I et F, et

libération des faces de l'objet dont la morphologie est changée ou affectée. C'est lors de la visualisation que sera créée à nouveau la structure des faces.

Primitive DETRUIRE (P);

Cas identique au précédent

4.2. Utilisation du lien objet/faces

Cette gestion dynamique du couple (objet,faces) au sein de la structure arborescente du système, permet d'envisager différemment les divers processus, en exploitant plus les possibilités du matériel.

4.2.1. Influence sur les processus de visualisation et description

* En ce qui concerne la visualisation, un gain de performances est obtenu puisque, de façon générale la décomposition en faces ne se fait qu'une fois à la première visualisation de l'objet.

* dans le domaine de la description, l'identification par désignation sur l'écran se voit facilitée. Le matériel, en effet, gère cette identification -par réticule par exemple- pour donner les numéros de faces, desquels est déduit l'objet identifié via le lien objet/faces.

En termes plus globaux, cette structure permet la manipulation d'objets d'une complexité quelconque, avec toute la puissance d'interactions que cela implique.

4.2.2. Influence sur les processus d'attribution et consultation

L'intérêt réside là dans l'exploitation de la mémorisation de certains attributs par le matériel -couleur ou texture, indicateur de visibilité, normale et repère de la face-. La structure de données principale en serait alors allégée.

Cela entraine bien sûr de nouveaux processus d'attribution et consultation -cette dernière se fait par matériel pour les attributs cités ci-dessus-, dont le guidage peut s'opérer en complètant la structure "TABLE" évoquée en paragraphe 3.1.2. de ce chapitre. S'y ajoute un tableau, indicé par le numéro de processus, indiquant si il y a ou non mémorisation au niveau matériel. L'élément ELTPROCESSUS devient alors :

```
ELTPROCESSUS = record
     TABCORRES : array (0..NBOBJLOG,1..NBOBJPHYS) of boolean;
     MEMORISATION : integer;
end;
```

5. Réalisations dans le cadre de l'unité de description-visualisation

Alors que jusque là nous nous étions attachés surtout à expliciter l'organisation des données et l'agencement des tâches, innovations essentielles, il est vrai dans ce logiciel, les lignes suivantes nous offrent l'occasion de décrire certains opérateurs plus élémentaires, mis en oeuvre justement par les unités déjà évoquées et plus spécialement l'unité de contrôle, opérateurs de visualisation et description réunis dans l'unité de même nom.

Parmi cette banque d'opérateurs, nous développerons les suivants :

- transformations géométriques "pures" -classe G-, avec ses techniques de base et ses règles de composition
- transformations géométriques d'affichage et de prise de vue -classes Ga et Gv-, et les problèmes de clipping
- description des informations de couleur
- description des informations d'éclairage

5.1. Les transformations géométriques

5.1.1. Utilisation de coordonnées homogènes

Le lecteur trouvera en ANNEXE 3 les éléments concernant cette technique, tirés de l'ouvrage rédigé par P.MORVAN et M.LUCAS traitant des techniques de base de l'infographie (MOL76).

5.1.2. Règles de composition

Afin de permettre une exploitation intéressante de la structure arborescente lors des compositions géométriques, le choix a été fait d'exprimer toute transformation géométrique d'un noeud par rapport au repère de son père, quelle que soit la position du père dans le repère global de la scène -ceci concerne donc l'attribution d'un attribut de type G2D ou G3D à un noeud de la structure-.

Par contre, en ce qui concerne la visualisation, le choix du repére peut être quelconque, et constitue un des critères du processus de visualisation.

Le calcul de la transformation géométrique que l'on doit appliquer au noeud à visualiser, pose alors quelques problèmes, illustrés par l'exemple suivant.

Soit une structure de données telle que la schèmatise la figure V.22.

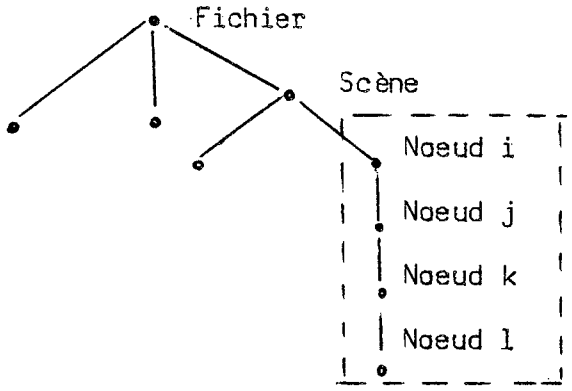


Figure V.22: Exemple de composition géométrique

Soit Gi, Gj, Gk et Gl les matrices définissant les positions des noeuds i, j, k et l par rapport au repère de leur père respectif.

Soit g la transformation géométrique à appliquer au noeud l, pour le visualiser dans le repère du noeud i.

g doit alors être exprimée dans le repère du noeud k, afin de pouvoir être composée avec Gl. Soit g2 sa transposition le permettant.

La géométrie du noeud l est alors transformée comme suit :

$$Gl2 = Gl \times g2$$

D'où $\quad Gl \times Gk \times Gj \times g = Gl2 \times Gk \times Gj$

$$= Gl \times g2 \times Gk \times Gj$$

Ces deux membres de cette équation expriment la composition des différentes géométries pour la visualisation de l par rapport à i. On en déduit g2 :

$$g2 = Gk \times Gj \times g \times Gk(-1) \times Gj(-1)$$

Ainsi sont nécessaires les transformations inverses de Gk et Gj, exprimant la position des noeuds pères de k et j dans les repères de k et j.

Dès lors, nous pouvons préciser d'une part la description de l'attribut de type Géométrie -cf.paragraphe 2.1.1. de ce chapitre-, ainsi que les procédures de calcul géométrique "CALCUL" et

"CALCUL-POINTS-COMPOSES" -cf.paragraphe 3.1.1. de ce chapitre-.

Pour ce qui est du descripteur d'attribut, deux matrices de transformation en coordonnées homogènes sont conservées:

- matrice de positionnement par rapport au noeud père

- matrice inverse

Ce qui donne le descripteur DGEOMETRIE suivant.

```
DGEOMETRIE = record
    TYPES : GEOMETRIE; (*G2D ou G3D*)
    VARICTE : integer;
    NBNOEUDSTOUCHES : integer;
    MAT : array (1..4,1..4) of real;
    MATINVERSE : array (1..4,1..4) of real;
end;
```

Pour ce qui est des procédures :

- "CALCUL" : compose dans une matrice de travail "MATRAV" les matrices "MAT" et "MATINVERSE" pour chacun des noeuds de contexte de visualisation

- "CALCUL-POINTS-COMPOSES" : compose les points de description de l'objet à visualiser avec la matrice "MATRAV"

Cette méthode nous permet ainsi de visualiser selon deux grands types de processus, ce que nous avions évoqué en paragraphe 2.2.4. du chapitre IV:

- visualisation "absolue", par rapport au repère global du fichier graphique

- visualisation "relative" par rapport au repère d'un noeud quelconque de la structure.

5.2. Fenêtre et cloture

Ces deux notions, dont nous donnerons dans un premier temps les éléments de base traduisent les géométries de point de vue et d'affichage, dont sera précisée la description de l'attribut correspondant, déjà entrevu en paragraphe 2.1.1. du chapitre V. Enfin, nous indiquerons l'algorithme choisi pour la phase de clipping en 2D.

5.2.1. Définitions de base

Deux espaces sont à considérer dans un logiciel graphique, chacun ayant un système de coordonnées propres :

- l'espace utilisateur, qui peut être lié aux périphériques de description -par exemple une tablette à digitaliser-

- l'espace écran, dépendant totalement du matériel utilisé

Ces espaces étant très disparates tant dans leur taille que dans leur précision -avec à chaque fois un avantage pour l'espace utilisateur-, deux outils de visualisation doivent être fournis :

- définition d'une fenêtre, correspondant à la classe d'informations Gv, permettant de n'afficher qu'une partie du dessin -partie visible dans la fenêtre- sur l'écran

- définition d'une cloture, correspondant à la classe d'informations Ga, allouant un certain espace de l'écran à une image, permettant par exemple d'afficher trois vues simultanées du même objet.

La figure V.23 illustre ces deux notions.

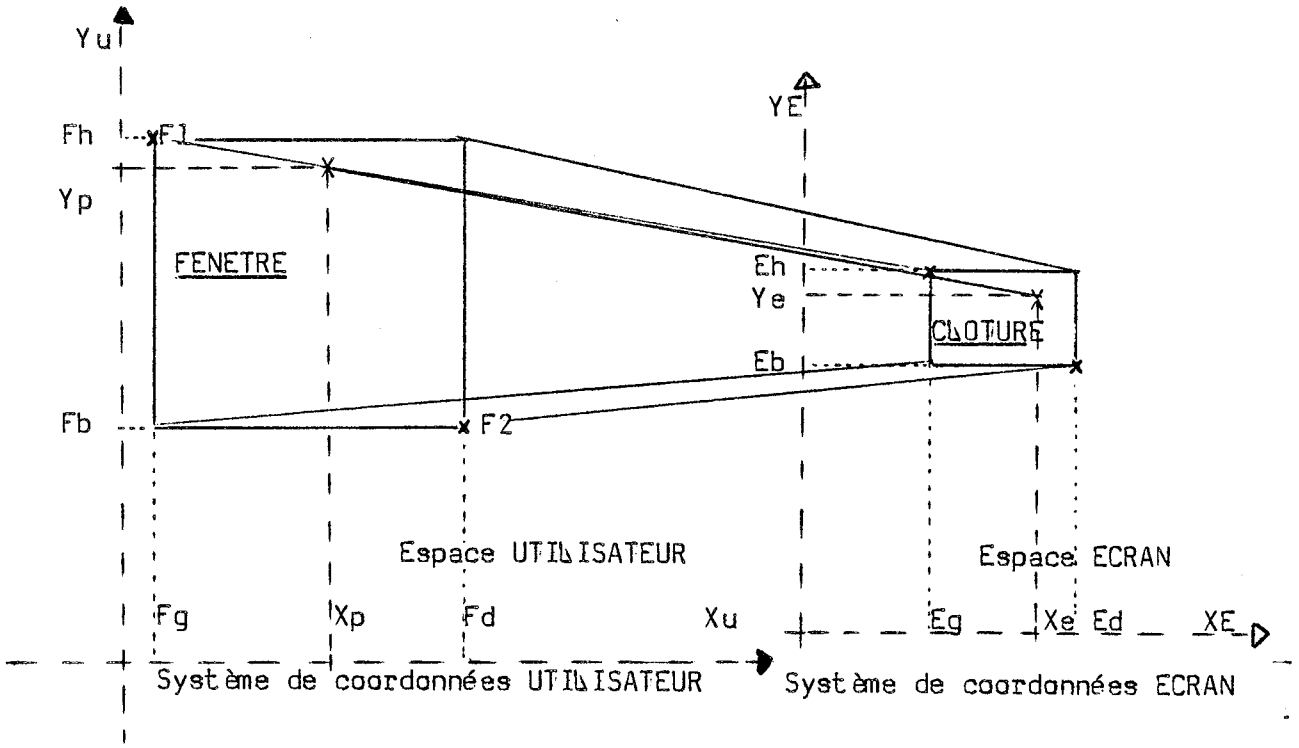


Figure V.23: Fenêtre et cloture

De ce schèma, ressort bien sûr la nécessité de passer des coordonnées utilisateur aux coordonnées écran -point (Xp,Yp) de la fenêtre au point (Xe,Ye) de la cloture-. Cette conversion est effectuée par la procédure "FENETREACLOTURE" évoquée en paragraphe 3.1.1. de ce chapitre.

En coordonnées utilisateur :

$$XFc = \frac{Fd + Fg}{2} \quad , \quad YFc = \frac{Fh + Fb}{2} \qquad : \text{centre fenêtre}$$

$$Hf = \frac{Fh - Fb}{2} \quad , \quad \Delta f = \frac{Fd - Fg}{2} \qquad : \text{hauteur et largeur fenêtre}$$

En coordonnées écran :

$$XEc = \frac{Ed + Eg}{2} \quad , \quad YEc = \frac{Eh + Eb}{2} \qquad : \text{centre cloture}$$

$$He = \frac{Eh - Eb}{2} \quad , \quad \Delta e = \frac{Ed - Eg}{2} \qquad : \text{hauteur et largeur cloture}$$

Ce qui donne les formules de passage d'un espace à l'autre suivantes :

$$Xe = \frac{Xp - XFc}{\Delta f} . \Delta e + XEc$$

$$Ye = \frac{Yp - YFc}{Hf} . He + YEc$$

5.2.2. Descripteur d'attribut

Pour des raisons de commodité, et devant la relative simplicité de ces deux notions en 2D, un seul descripteur d'attribut, regroupant les classes Ga et Gv, a été introduit. Voici sa description :

```
DGEOMGVA = record
    TYPES : TYPGVA;
    VARICTE : integer;
    NBNOEUDSTOUCHES : integer;
    XΔW,YBW,XRW,YTW : real;
    XΔV,YBV,XRV,YTV : real;
end;
```

avec (XBW,YBW) : coin inférieur gauche cloture;
(XRW,YTW) : coin supérieur droit cloture;
(XBV,YBV) : coin inférieur gauche fenêtre;
(XRV,YTV) : coin supérieur droit fenetre;

5.2.3. Le clipping en 2D

Le second problème qui se pose, après celui de passage des coordonnées d'un espace à l'autre, est celui de calculer les parties visibles du dessin, déterminées par les limites de la fenêtre.

L'algorithme de clipping -ou coupage- que nous présentons, s'applique essentiellement à des segments de droite. Ainsi, avant application de cet algorithme, les objets à visualiser sont décomposés en segments.

Le principe de l'algorithme est le suivant :

- l'espace utilisateur est découpé en 9 régions, en prolongeant les bords de la fenêtre -cf.figure V.24-.

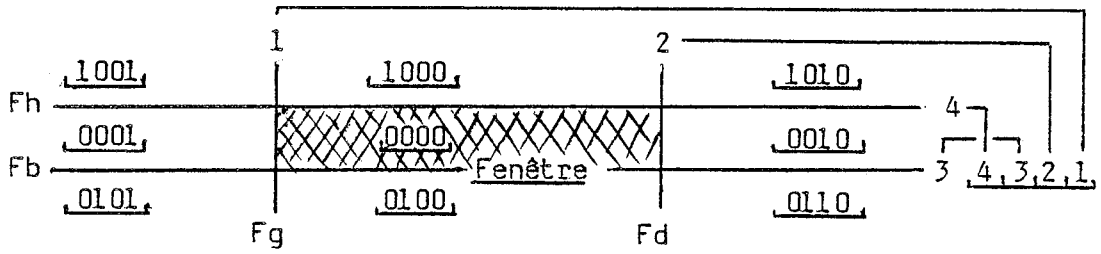


Figure V.24: Découpage et codage de l'espace utilisateur

- chaque région est codée sur 4 bits, chacun des bits représentant la position de la région par rapport à la droite correspondante délimitant la fenêtre et l'intérieur de la fenêtre : 0 du même coté, 1 de l'autre coté. On obtient ainsi le codage représenté en figure V.24.

L'algorithme consiste à :

- associer aux extrémités de chaque segment le codage du plan auquel elles appartiennent
- faire l'intersection des deux codes binaires du segment
- si le résultat est non nul : segment non visible
- si le résultat est nul, on divise le segment en 2, et on applique le processus jusqu'à trouver un segment visible, c'est-à-dire dont les deux codes sont (0000).

Un exemple est donné en figure V.25. Les parties visibles des segments sont représentées en trait plein, les parties non visibles en pointillé. Le chiffre associé à chaque segment indique le nombre d'itérations (tests + calculs d'intersection).

Il faut noter que cet algorithme peut être cablé.

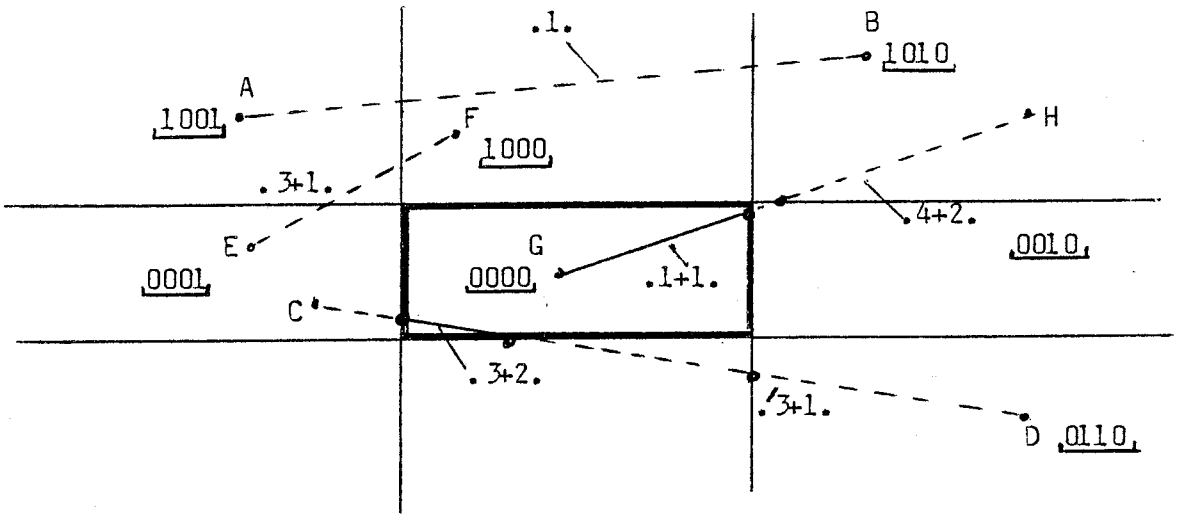


Figure V.25: Exemple d'algorithme de coupage

La réalisation du clipping est faite dans la procédure "CLIPPING2D" au sein du module "DESATTS", dans le cadre de l'agencement interne des processus de visualisation, rôle dévolu à l'unité de contrôle -cf. paragraphe 3.1.1. de ce chapitre-.

5.3. Description d'une couleur

Le but de ce processus est de permettre une description interactive d'une couleur, sans référence à sa modèlisation interne. La réalisation de ce processus se heurte à deux problèmes :

- HELIOS permettant une gamme de 4096 couleurs différentes, il est impossible de toutes les visualiser simultanément

- une présentation harmonieuse de toutes les nuances n'est guère possible, étant donné la nature tridimensionnelle des espaces de couleurs.

Nous avons choisi de décomposer ce processus en 3 phases :

- affichage d'une palette réduite de couleurs

- identification par désignation d'une couleur sur cette palette

- description interactive de la couleur souhaitée à partir de cette couleur de base

Notons, qu'à ces 3 phases correspondent 3 primitives utilisateur, déjà énoncées en paragraphe 2.2.4. du chapitre IV.

5.3.1. Affichage d'une palette

Devant l'impossibilité d'afficher toutes les nuances permises par HELIOS, force a été de restreindre cet affichage à une palette de 240 couleurs , que nous pouvons appeler "de base", exprimées en (R,V,B).

Les paramètres d'affichage de cette palette sont les suivants :

- définition d'une cloture -centre(xc,yc) et coefficients d'échelle (zxc,zyc)- permettant ainsi d'afficher simultanément la scène et la palette dans un "coin" de l'écran

- détermination de la couleur de départ et du pas d'incrémentation choisis pour le degré de dégradé des couleurs de la palette. Par défaut, ces deux paramètres sont à 15, ce qui donne une palette allant du bleu au jaune, en passant par le mauve, rouge et vert.

- booléen indiquant si il s'agit d'une nouvelle palette ou d'une palette déjà visualisée -dans ce cas, le temps d'affichage est immédiat-.

Cette tâche, réalisée par le module "COMVAX" et invoquée par une commande HELIOS dans le module "HELPILOT", exploite les possibilités d'utilisation sur HELIOS de plusieurs plans. Le prototype du laboratoire permettant 2 plans, nous avons opté pour l'organisation suivante :

- plan 0 : plan d'identification -donc incluant et gèrant les faces HELIOS- sur lequel est visualisée la scène composée des objets.

- plan 5 : plan de textures -ici des couleurs- mémorisant la palette à afficher

Ainsi sont permis, d'une part l'affichage simultané de ces deux plans - le plan 0 ayant automatiquement la priorité sur le plan 5-, d'autre part un effacement sélectif du seul plan 5, facilitant ainsi la tâche de l'utilisateur -cf.figure V.26-.

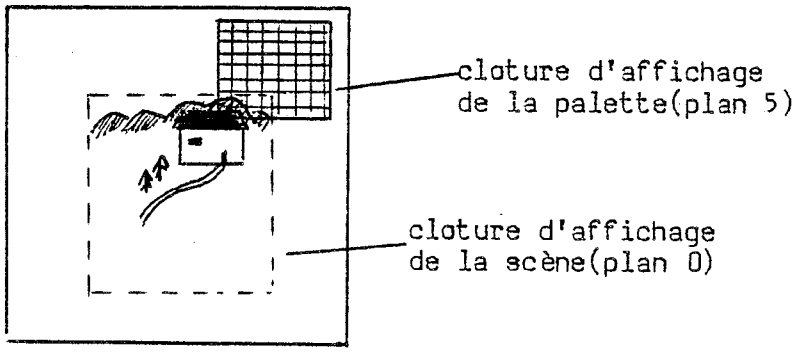


Figure V.26: Affichage simultané d'une scène et de la palette de couleurs

5.3.2. Identification d'une couleur de base

Cette primitive permet à l'utilisateur, par désignation sur l'écran au moyen du réticule, de choisir une couleur exprimée en (R,V,B) comme couleur de base de sa description.

Les coordonnées (x,y) données en retour de l'identification via le réticule par le matériel, permettent de récupérer au niveau logiciel la couleur (R,V,B) par recherche dans une table créée au moment de l'affichage de la palette -cf.figure V.27-.

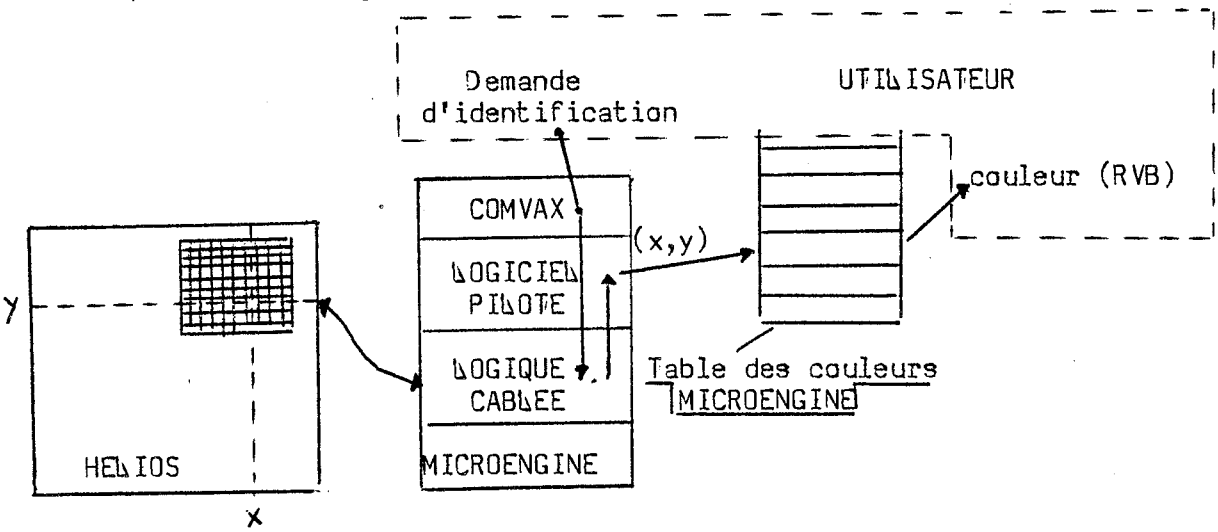


Figure V.27: Identification d'une couleur de base

5.3.3. Description de la couleur finale

Le but de cette primitive est la description interactive -avec visualisation à chaque étape de description- de la couleur finale souhaitée par l' utilisateur à partir de la couleur de base identifiée sur la palette.

Cette phase n'est pas possible dans l'espace (R,V,B), l'utilisateur ne connaissant pas -et n'ayant pas à connaitre- la modèlisation interne d'une couleur. Le travail est réalisé dans l'espace (T,L,S) dont les paramètres sont plus facilement utilisables -cf.ANNEXE 4-.

- T (Teinte) : vers le Rouge, Vert ou Bleu : à partir des valeurs de base -respectivement -2pi/3, 0 et +2pi/3-, chaque invocation de l'option incrémente ou décrémente T de pi/100.

- L (Luminance) : + ou - à partir d'une valeur de base de 5, incrémentation ou décrémentation de 1.

- S (Saturation) : + ou - à partir d'une valeur de base de 15 0/0, l'option incrémentant ou décrémentant de 1 0/0.

Ainsi, à chaque pas de description via les différentes options de ces 3 paramètres, l'utilisateur peut choisir de visu la couleur qui l'intéresse.

Cette description nécessite une conversion des couleurs de l'espace (RVB) vers l'espace (TLS) et réciproquement. Les expressions suivantes indiquent les règles de cette conversion :

$$T = \arcsin (\sqrt{3/2} * (V-R) / S)$$
$$L = (R + V + B) / 3$$
$$S = \sqrt{R^{**}2 + V^{**}2 + B^{**}2 - B*V - B*R - V*R}$$

$$R = L - 1/3 * S * \cos T - 1/\sqrt{3} * S * \sin T$$
$$V = L + 2/3 * S * \cos T$$
$$B = L - 1/3 * S * \cos T + 1/\sqrt{3} * S * \sin T$$

La primitive de description, implantée dans le module "COMVAX", et invoquée sous forme de commande dans le module "HELPILOT", se présente alors sous la suite d'actions suivantes, présentées ci-après :

* Remarques:

. Le fait de travailler sur le plan 0, plan d'identification donc, pour tout ce qui touche à la visualisation de la scène, impose de transmettre au matériel non une couleur codée en (R,V,B), mais un pointeur de couleur , adressant une zone de la mémoire de textures -en l'occurence le plan 5- -cf.parag. 2.4.2.du chapitre III-. Aussi, après description de la couleur, celle-ci est rangée dans une zone de la mémoire de textures correspondant à un pointeur, pouvant aller du no. 480 au no. 495. Ces 16 pointeurs adressent en fait la dernière ligne de l'écran -la gestion des pointeurs de couleurs ou textures gère un lien étroit entre le numéro de pointeur et la position ainsi que la taille de la texture associée-, découpée donc en 16 zones de couleur d'une taille de 16x16 pixels. Ce qui revient à dire que 16 couleurs sont descriptibles simultanément, nombre que l'on peut étendre.

. La description que nous venons d'expliciter est ce que nous avions dénommé en paragraphe 3.3.2. du chapitre III une description explicite par construction. La description de la couleur peut se faire également par référence par le processus suivant :

- identification d'un objet de la scène

- consultation de son attribut de couleur

- description, à partir de cette couleur de base, d'une couleur finale par le biais de la primitive explicitée ci-dessus.

```
Procedure DESCRIPTION-COULEUR (COUL-BASE; var COUL-FINALE);
begin
   CONVERS-TLS (COUL-BASE);
```

```
case CHOIX of
   T:(*Teinte*)begin
      tb = 2pi/3; tr = -2pi/3; tv = 0;
      case CHOIX2 of
         R:(*Rouge*)begin
            t = tr - pi/100;
            CONVERS-RVB (COUL-FINALE);
            tr = t;
         end;
         V:(*Vert*)begin
            t = tv + pi/100;
            CONVERS-RVB (COUL-FINALE);
            tv = t;
         end;
         B:(*Bleu*)begin
            t = tb + pi/100;
            CONVERS-RVB (COUL-FINALE);
            tb = t;
         end;
      end;
      L:(*Luminance*)begin
         l = 5;
         case CHOIX2 of
            '+':begin
               l = l + 1;
               CONVERS-RVB (COUL-FINALE);
            end;
            '-':begin
               l = l - 1;
               CONVERS-RVB (COUL-FINALE);
            end;
         end;
      end;
      S:(*Saturation*)begin
         s = 15;
         case CHOIX2 of
            '+':begin
               s = s + 1;
               CONVERS-RVB (COUL-FINALE);
            end;
            '-':begin
               s = s - 1;
               CONVERS-RVB (COUL-FINALE);
            end;
         end;
      end;
   end;
```

5.4. Description de l'éclairage

De même que pour la couleur, il s'agit là d'une description interactive, visualisant à chaque pas de description les attributs ainsi décrits.

Cette description porte sur les trois informations que comporte la classe "E" -cf.paragraphe 2.1.1. de ce chapitre-, à savoir :

- couleur de la source lumineuse
- direction de la source lumineuse en coordonnées polaires
- intensité de la lumière ambiante

A chacun de ces types d'attributs correspond tout naturellement une description pouvant être invoquée par l'utilisateur.

Si aucune description n'est donnée, le logiciel prend en compte les valeurs standard suivantes :

- 4095, soit une source lumineuse blanche
- des angles de 0 et pi/2, soit une direction normale à l'écran
- une lumière ambiante moyenne de 5

5.4.1. Description de la couleur de la source lumineuse

Elle est strictement identique à celle présentée dans le chapitre précédent. Simple différence, le résultat exploité et communiqué au matériel est la valeur exprimée en (R,V,B) et non un pointeur de couleur.

5.4.2. Description de la direction de la source

Elle consiste à décrire interactivement les deux angles a et b, constituant les coordonnées polaires de cette direction -cf.figure V.28-.

Nous avons choisi de faire exprimer ces deux angles de facon simple et appréhensible par l'utilisateur :

- vers le Haut, vers le Bas : correspondant à b.
- vers la Droite, vers la Gauche : correspondant à a.

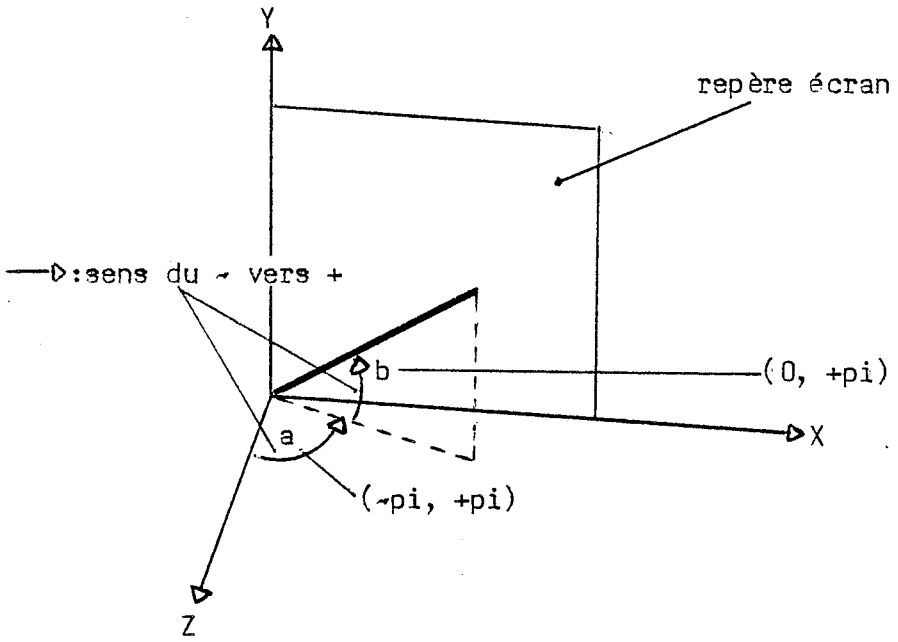


Figure V.28: Direction exprimée en coordonnées polaires / repère écran

```
        Procedure DESCRIPTION-ANGLES (var A,B:real);
begin
   As = 0;
   Bs = pi/2; (*valeurs standard*)
  case CHOIX of
   H:(*Haut*)begin
      B = Bs + pi/60;
      Bs = B;
   end;
   B:(*Bas)begin
      B = Bs - pi/60;
      Bs = b;
   end;
   D:(*Droite*)begin
      A = As + pi/60;
      As = A;
   end;
   G:(*Gauche*)begin
      A = As- pi/60;
      As = A;
   end;
  end;
end;
```

5.4.3. Description de la lumière ambiante

Cette information, exprimant la proportion de lumière diffuse dans l'atmosphère et éclairant en particulier les parties d'objets non soumises aux rayons de la source lumineuse, peut prendre 16 valeurs -pas de lumière ambiante à 0, maximale à 15-.

L'utilisateur décrit donc cet attribut par invocations de "+" ou "-" de lumière ambiante.

```
Procedure DESCRIPTION-LUMAMB(var L:integer);
begin
   Ls = 5 ;(*valeur standard*)
   case CHOIX of
      '+':begin
         L = Ls + 1;
         Ls = L;
      end;
      '-':begin
         L = Ls - 1;
         Ls = L;
      end;
   end;
end;
```

6. Exemple d'utilisation du logiciel

Un exemple d'utilisation documenté et illustré du logiciel à partir d'un programme d'application est donné en ANNEXE 5.

CONCLUSION

Le logiciel, dans sa version 2D, et tel qu'il a été décrit tout au long des pages de ce mémoire, est à présent terminé. Sa réalisation s'est faite en parallèle avec celle destinée au matériel TEKTRONIX 4114, validant ainsi l'option multi-applications et multi-matériels.

La mise en oeuvre s'est donc heurtée aux inévitables problèmes de cohérence de versions et de synchronisation, résolus par de fréquentes discussions au sein du groupe de travail.

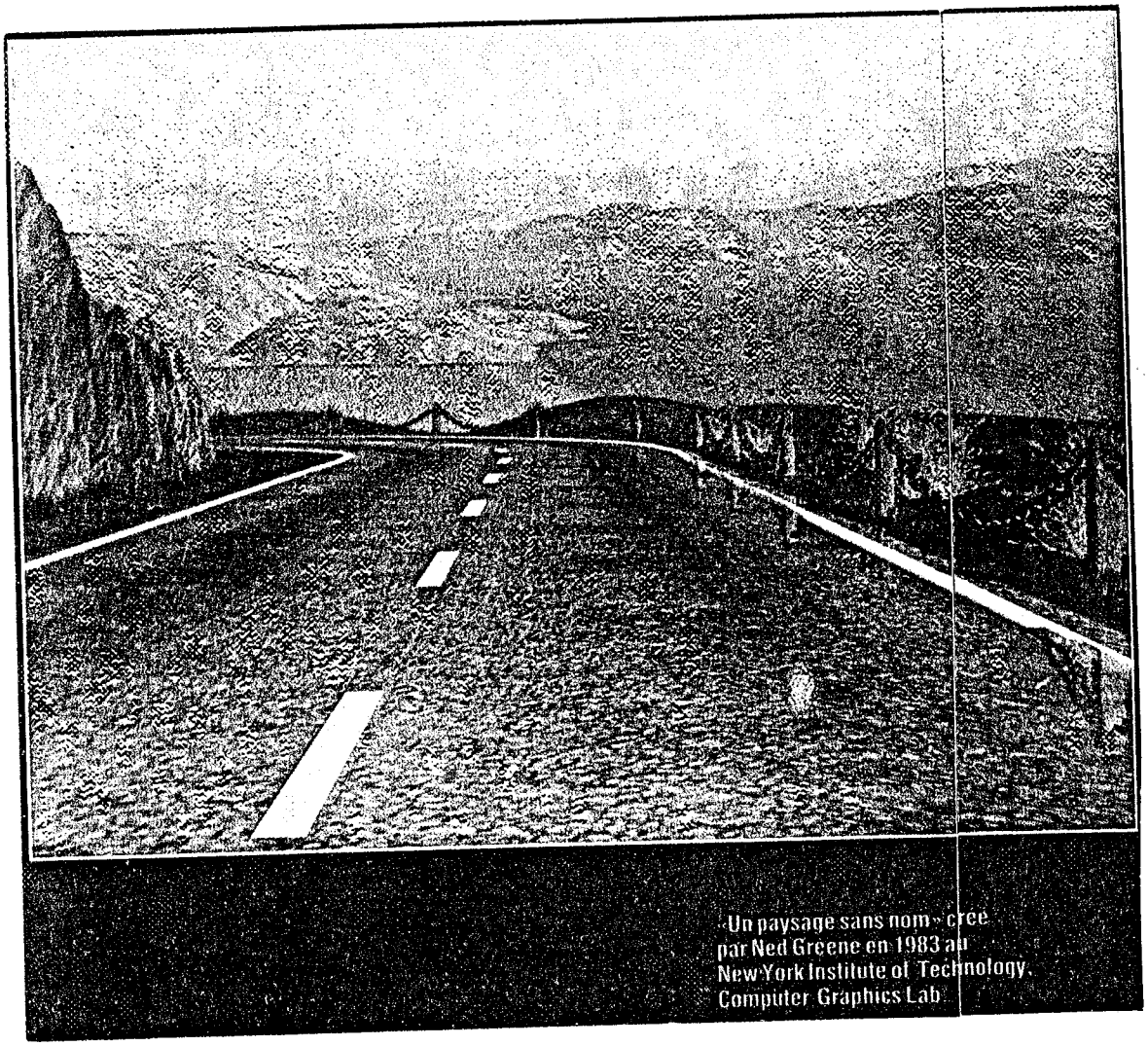
Deux voies d'avenir me semblent se dégager pour ce logiciel, arrivé aujourd'hui à ce que l'on peut appeler un bon produit de base, structuré et tout à fait opérationnel.

La première voie se situe dans le domaine "Recherche et développement" avec la réalisation d'une version 3D, avec tout ce que cela entraine au niveau de la modèlisation et des problèmes d'analyse de visibilité. Cette phase est réalisable à court terme, puisque s'y est déjà attelé P.GENOUD, dans le cadre d'une thèse de 3e. cycle, auteur d'un document de travail interne à l'équipe portant sur la modèlisation tridimensionnelle (GEN84).

Toujours dans ce même domaine de recherche, peut être envisagé, me semble-t-il, la conception du logiciel pilote du nouveau prototype d'HELIOS à architecture banalisée, que nous avons déjà évoqué.

La seconde voie d'avenir est celle de la commercialisation -envisagée avec la société I.T.M.I.- autour d'un concept quelque peu différent, à savoir celui de "workstation" -ou station de travail-. Cette formule, adoptée par tous les constructeurs et spécialement aux USA, concoit le terminal comme un ensemble très évolué, muni d'un microcalculateur -CPU à mots de 16 bits en général- gèrant un environnement classique -disques durs à grosse capacité, I/O avec périphériques classiques, ...- outre bien sûr le contexte purement graphique -avec en particulier la base de données graphiques-. Cette voie a donc déjà commencé à être explorée, permettant un transfert de technologie exemplaire entre la recherche et l'industrie.

Nous terminerons ce mémoire par une image -tirée du numéro spécial -de "Sciences et Techniques" du mois de mai 1984-, symbolisant à la fois les résultats permis par la synthèse d'images mais aussi la voie parsemée d'embûches -gare à la route sinueuse et glissante ...- permettant d' obtenir ce fantastique réalisme.



« Un paysage sans nom » créé par Ned Greene en 1983 au New York Institute of Technology, Computer Graphics Lab

ANNEXES

ANNEXE 1 : VERS UNE ARCHITECTURE BANALISEE DES SYNTHETISEURS D'IMAGES

PREMIER COLLOQUE IMAGE
Traitement, Synthèse, Technologie et Applications
BIARRITZ — Mai 1984 —

VERS UNE ARCHITECTURE BANALISÉE DES SYNTHÉTISEURS D'IMAGE

FRANCIS MARTINEZ

Institut IMAG / Laboratoire ARTEMIS / B.P. 68 / 38402 SAINT MARTIN D'HERES cédex

RESUME

Chaque semaine un nouveau terminal de synthèse d'image fait son apparition sur le marché et l'utilisateur a de plus en plus de difficultés à déterminer celui qui convient le mieux à ses applications. Deux types de configurations sont généralement opposées :

- Un terminal "bas de gamme" muni de logiciels permettant de satisfaire un grand nombre d'applications sans aucune possibilité d'interaction temps réel.
- Un terminal évolué adapté à une classe spécifique d'applications et proposant des fonctions temps-réel. Dans ce dernier cas l'architecture du synthétiseur fait appel à un nombre important d'unités de traitement cablées et la reconfiguration du processus interne du terminal est quasiment impossible.

L'architecture modulaire présentée dans cet article tente de résoudre ces problèmes en considérant séparément les informations de morphologie, de géométrie, d'aspect et d'éclairage nécessaires à la définition d'une image. Ces informations sont traitées et synthétisées par des unités de traitement banalisées, connectées en fond de panier sur un bus standardisé. Toutes ces unités sont cablées et opèrent à la vitesse du signal vidéo.

Cette architecture originale permet de nombreuses combinaisons permettant de répondre à une grande variété d'applications. Un prototype offrant des possibilités en temps réel (apposition de textures et calcul d'éclairage) est actuellement en cours de réalisation à l'Université de Grenoble.

SUMMARY

Every week a new raster graphic device is available and the user has to select the best one according to his applications. This choice is not easy and two kinds of configuration generally face him.

- A low-cost terminal with a lot of software packages which satisfy many applications but doesn't allow any real-time interaction.
- A powerful terminal based on a specific application which allows many real-time facilities. In this case, the architecture of the synthetiser uses lot of wired units and modifications of the inner process is quite impossible.

The modular architecture discussed in this paper attemps to resolve these problems by separating morphology, geometry, aspect and lighting information which are handled apart by general purpose process units plugged on a standardized bus. All this units work at video rate.

This original architecture allows many differents combinations according to the application requirements. A prototype-system with some real-time capabilities (texturing and lighting process) is currently developped at the University of Grenoble.

VERS UNE ARCHITECTURE BANALISÉE DES SYNTHÉTISEURS D'IMAGE

Francis MARTINEZ

1. PRESENTATION

La mutation technologique de ces dernières années ainsi que l'accroissement des besoins en images synthétiques conduisent à une remise en cause périodique de l'architecture des terminaux dédiés à la synthèse d'images. La tendance actuelle est de répondre à la grande variété des performances exigées par une aussi grande variété de machines fondées sur des architectures spécialisées. L'éventail des possibilités en matière de qualité d'image (allant du dessin au trait aux images réalistes) ou encore en matière de dynamisme (allant de l'image entièrement statique à l'animation en temps réel) est ainsi pratiquement couvert par la gamme des terminaux disponibles.

Cependant, cette spécialisation conduit inévitablement à une diversification néfaste sous plusieurs aspects. En premier lieu, elle restreint notablement le champ d'application de chaque machine de synthèse, limitée par les possibilités d'une architecture figée et les performances "actuelles" de la technologie utilisée. En second lieu, la disparité et la spécificité des techniques entraîne une incompatibilité totale entre les différents terminaux, et l'absence de possibilités d'extension et d'adaptation enchaîne le plus souvent l'utilisateur à un constructeur voire même à un terminal donné.

Face à cette situation nous proposons de jeter les bases d'une architecture "universelle" de synthétiseur d'image, permettant un haut degré de parallélisme, extensible et modulaire afin de répondre par simple programmation ou adjonction de modules, aux besoins spécifiques d'applications variées. Cette architecture supporte aussi bien des terminaux standards pour lesquels la couleur de chaque pixel est enregistrée dans une mémoire de trame, que des synthétiseurs temps-réel d'images réalistes prenant en charge, en parallèle, le calcul cablé de textures colorées, de l'éclairement d'objets tri-dimensionnels et de l'élimination des parties cachées.

Après avoir présenté les principes de base de la synthèse d'image, nous en déduirons les fondements d'une telle architecture, basée notamment sur la séparation et la "parallélisation" des informations graphiques.

2. LES CONCEPTS LIES A LA SYNTHESE D'UNE IMAGE.

Ce paragraphe est destiné à mettre en valeur les différents éléments nécessaires à la synthèse d'une image. Nous serons ainsi amenés à considérer successivement :
- les informations traitées,
- les opérateurs les concernant,
- les processus permettant de combiner ces opérateurs.

2.1. Les classes d'information

Les informations véhiculées par une image peuvent être partitionnées en six classes indépendantes [Mar 82]:

Identité	:	le nom des objets
Morphologie	:	leur forme intrinsèque
Aspect	:	leur aspect intrinsèque (couleur, texture)
Géométrie	:	leur situation dans la scène
Eclairage	:	leur réaction à la lumière (réflexions)
Structure	:	leurs liens hiérarchiques ou autres.

2.2. Les opérateurs élémentaires de synthèse

La synthèse d'image consiste à combiner les six classes d'informations initiales ci-dessus pour obtenir l'information finale exprimant la couleur en chacun des pixels de l'image (classe A).

Pour effectuer cette synthèse il faut procéder par étapes successives à l'aide d'opérateurs élémentaires de calcul tels que ceux présentés ci-après [Mar 82].

Synthèse M' = f(M,G) :

Application d'une transformation géométrique à une forme (M).

Synthèse A' = f(A,E) :

Modulation de l'aspect en fonction de l'éclairage.

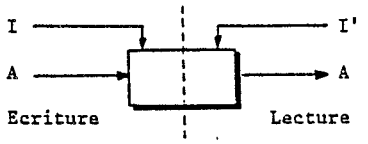
Correspondance A = f(I) :

Délivre un aspect en fonction de l'identité.

Tous ces opérateurs sont des opérateurs de calcul synchrones ce qui signifie que les résultats sont exploités dès que les données nécessaires ont été fournies.

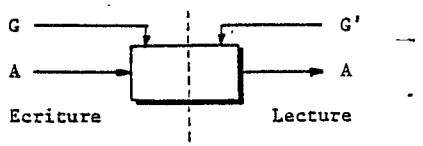
Cependant, la synthèse d'une image nécessite au moins un opérateur asynchrone servant de "tampon" entre deux processeurs différents (par exemple la logique cablée et le micro-processeur pilote). Cet asynchronisme est assuré généralement par un *opérateur de mémorisation* tel que ceux présentés ci-après.

Correspondance A = f(I) :



Cet opérateur est une simple "table de couleurs" délivrant une couleur (A) pour chaque objet exprimé à travers son identité (I) .

Correspondance A = f(G)



Il s'agit ici d'une mémoire de trame (mémoire d'image) délivrant une couleur (A) pour chaque pixel, exprimé par sa position (G) .

2.3. Les processus de synthèse d'image

Les performances d'un synthétiseur d'image (rapidité, qualité et complexité des images) dépendent directement du choix des opérateurs élémentaires et de leur ordonnancement. Nous avons montré lors de travaux précédents [Mar 82] que cet ordonnancement peut être déduit d'une façon quasi-systématique des performances requises. On peut illustrer ces propos en s'appuyant sur le processus d'un terminal "classique" schématisé sur la figure 1.

VERS UNE ARCHITECTURE BANALISÉE DES SYNTHÉTISEURS D'IMAGE

Francis MARTINEZ

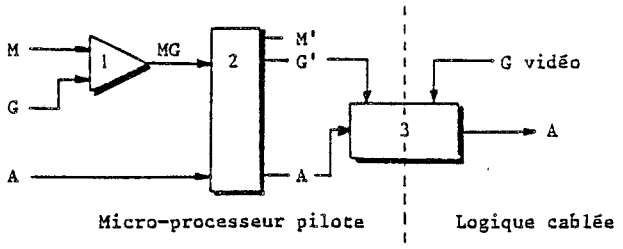


Figure 1

Si ce terminal traite des taches polygonales uniformes, le rôle des opérateurs est le suivant :

Opérateur 1 : Application des transformations géométriques (translations, rotations, homothéties) aux sommets de chaque polygone. Le résultat est un nouveau polygone.

Opérateur 2 : Décomposition du polygone en pixels. L'objet élémentaire considéré à présent est donc *le pixel* pour lequel la morphologie (M') est implicite, la géométrie (G') exprime les coordonnées et l'aspect (A) est celui du polygone initial.

Opérateur 3 : Il s'agit de la mémoire de trame dans laquelle est mémorisée la couleur de chaque pixel.

A partir de ce cas très simple, on peut envisager une infinité de variantes adaptés aux besoins spécifiques des applications. Par exemple si l'on désire favoriser l'interactivité temps réel au niveau des couleurs on ajoute "après" la mémoire de trame une table de correspondance (Opérateur 4 de la Figure 2).

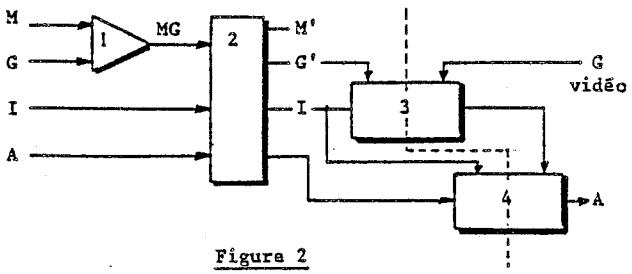


Figure 2

Si l'on désire présenter des surfaces gauches comportant toutes les nuances dues à l'éclairage tout en conservant l'interactivité temps réel sur les couleurs on peut enregistrer dans une seconde mémoire de trame (opérateur 5) les informations d'éclairage relatives à chaque pixel (réflexion diffuse et spéculaire). Le processus est alors celui de la figure 3 dans lequel l'opérateur 6 effectue la modulation de la couleur en fonction du modèle d'éclairage.

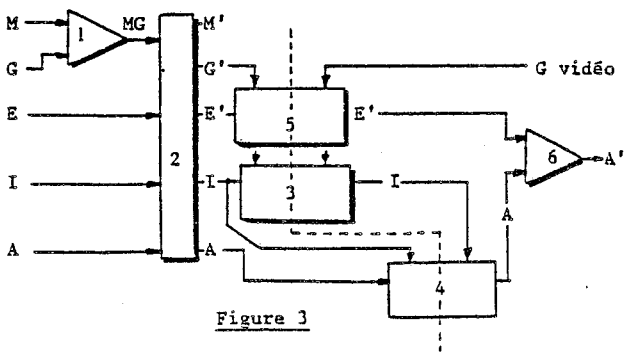


Figure 3

3. INCIDENCE AU NIVEAU DE L'ARCHITECTURE D'UN SYNTHETISEUR

Les considérations évoquées ci-dessus ont une incidence directe sur l'architecture interne d'un synthétiseur d'image. Nous nous intéresserons ici plus particulièrement à la partie cablée qui conditionne très largement les performances d'ensemble.

3.1. La nature des modules

Nous nommerons *module* un processeur cablé (généralement regroupé sur une carte) associé à un opérateur élémentaire du processus. Outre les modules de calcul, les opérateurs de mémorisation revêtent une forme différente selon que l'information qui joue le rôle d'adresse est mono-dimensionnelle (cas des tables de correspondance) ou bi-dimensionnelle (cas des mémoires de trame). Trois types de modules peuvent donc être présents dans un système. A ceci il convient cependant d'ajouter deux modules spéciaux extérieurs au procesus de synthèse : le *module de communication* chargé de l'écriture des mémoires et de la configuration des calculs ; le *module d'affichage vidéo* chargé de la génération du signal vidéo. Tous ces modules doivent être capables de fonctionner à la vitesse du signal vidéo soit dans une période de 75 ns environ dans le cas du standard T.V. Si le traitement à effectuer dépasse le temps imparti (cas des mémoires de trame et des calculs complexes) une architecture pipe-line permet de décomposer ce traitement en N sous-espaces. Le retard introduit par le module est alors de N cycles.

3.2. La nature des bus

Les bus internes du synthétiseur cablé véhiculent toutes les informations présentes dans le processus de synthèse. Ainsi, si l'on se reporte à la figure 3, la partie cablée comporterait 5 bus véhiculant en synchronisme les informations :

G vidéo, E', I, A, et A').

Le débit de ces bus est également celui du signal vidéo et leur "largeur" dépend de l'information transmise soit, par exemple 20 bits pour G vidéo (1024×1024), 8 bits pour E' (intensité), 12 bits pour I (numéro d'objet), 18 bits pour A et A' (6 bits par composante RVB).

Il faut, en outre, prévoir l'écriture des informations dans les modules de mémorisation. Si la partie micro-programmée comporte plusieurs processeurs pilotes indépendants fonctionnant en parallèle, il est intéressant de prévoir plusieurs bus d'écriture. Dans le cas où tous les opérateurs micro-programmés sont confiés à un seul micro-processeur, un seul bus de données est nécessaire. Ce bus doit avoir la largeur de la plus grande de toutes les informations à transmettre. Certaines informations jouent le rôle d'adresse (G' et I dans le processus de la figure 3) et la largeur du bus d'adresse est également celle de la plus grande de ces informations.

3.3. La synchronisation du processus

Si l'on se reporte encore une fois au processus de la figure 3, il est clair que les informations (A) et (E') doivent parvenir en synchronisme aux entrées de l'opérateur 6. Or les chemins parcourus sont différents et les opérateurs traversés peuvent induire des retards également différents. Il est donc indispensable d'introduire dans le plus court chemin un opérateur neutre introduisant un retard de compensation.

Francis MARTINEZ

4. LA BANALISATION DE L'ARCHITECTURE

Les terminaux actuels sont généralement conçus selon un processus donné et obéissent donc aux principes ci-dessus. Ces terminaux sont parfaitement adaptés à un type d'application mais deviennent inéfficaces voire même inutilisables lorsque l'on désire les introduire dans un processus différent de celui pour lequel ils ont été conçus. Ainsi si la mémoire de trame est destinée à recevoir la couleur de chaque pixel, on ne peut l'utiliser pour enregistrer un numéro d'objet, ou une profondeur pour l'algorithme du Z-buffer, ou encore un vecteur normal pour les surfaces gauches.

Notre objectif est de proposer une architecture modulaire, banalisée, et permettant une reconfiguration complète de la logique cablée en vue de s'adapter à une majorité de processus de synthèse. Nous exposons ci-après les problèmes posés par ce type d'approche et les solutions qui ont été retenues pour la conception d'un prototype en cours de réalisation au Laboratoire ARTEMIS de l'Institut IMAG de Grenoble. Nous reprendrons successivement les trois points évoqués ci-dessus : les modules les bus et enfin la configuration et la synchronisation du processus.

4.1. Les modules banalisés

Il est clair que la banalisation de l'architecture passe d'abord par celle des modules : les mémoires de trame, les tables de correspondances et les modules de calcul. Si les derniers restent spécifiques à un opérateur donné, les deux premiers peuvent mémoriser des informations quelconques et un type unique de mémoire de trame et de table de correspondance a été développé. Chaque module, quel qu'il soit, fait actuellement l'objet d'une carte "double europe" et obéit au même schéma de connexion sur le fond de panier.

4.2. Les bus banalisés

Le nombre et la largeur des bus à prévoir dépend des processus que devra supporter le synthétiseur. Il est bien entendu que la plus grande souplesse est donnée par un grand nombre de bus les plus larges possibles. Par ailleurs tous ces bus doivent être situés dans le fond de panier afin de pouvoir alimenter ou collecter tous les modules présents. En appliquant strictement ces principes un synthétiseur banalisé susceptible d'être configuré selon des processus variés devrait comporter en fond de panier un nombre de lignes prohibitif (environ 180 lignes pour le processus simple de la figure 3).

Une première remarque permet de constater que le bus "G vidéo" ne s'adresse qu'aux mémoires de trame et qu'il peut être supprimé si l'on introduit dans chaque mémoire de trame les compteurs (X,Y) effectuant le balayage. Cette option permet en outre d'effectuer la translation cablée des différentes mémoires.

En second lieu l'expérience permet de vérifier qu'un grand nombre de bus réduits est plus intéressant qu'un nombre limité de bus plus larges. Il est alors possible, pour les applications exigeant une plus grande précision, de coupler deux bus pour véhiculer l'information.

Le test d'un grand nombre de configurations répondant à des besoins divers a montré qu'une largeur de 12 bits est un bon compromis et que le nombre de bus peut être limité à 8 ; soit 96 lignes en tout. Ces lignes occupent la partie basse du fond de panier qui est schématisé sur la figure 4.

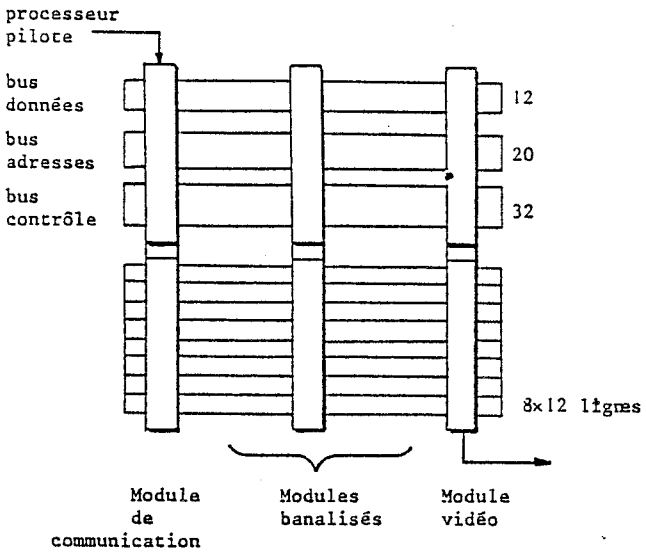


Figure 4

Compte-tenu des données ci-dessus les modules doivent répondre aux spécifications ci-après :

- Mémoire de trame : plan de 1024 × 1024 × 12 bits, la sortie peut être aiguillée sur l'un quelconque des 8 bus.
- Table de correspondance : mémoire de 4096 × 12 bits, l'entrée et la sortie peuvent être aiguillées sur l'un quelconque des 8 bus.

4.3. Configuration et synchronisation du processus

La source et la destination de chaque module sont programmées selon le processus à suivre. Chaque module contient, a cet effet, un registre indiquant quels sont les bus concernés. Ce registre peut être modifié à tout moment par le processeur pilote. Le logiciel doit, par conséquent, être capable de gérer l'occupation des bus pour l'ensemble des processus envisageables. Cette gestion soulève un certain nombre de problèmes délicats et constitue l'un de nos axes de recherche actuels.

Un autre problème crucial est celui de la synchronisation des informations. Il n'est plus possible ici d'introduire un module de retard fixe puisque la durée et la localisation de ce retard dépendent du processus de synthèse. Nous avons résolu ce problème en associant à chaque bus une ligne indiquant la présence de données valides sur le bus ce qui correspond à la partie visible d'une ligne vidéo. Ce signal est répercuté sur le bus de sortie d'un module au même titre que les données et avec le même retard. Lorsqu'un module de calcul nécessite la synchronisation de deux informations on adjoint à chaque bus d'entrée une file d'attente asynchrone intégrée dont le remplissage est commandé par le signal de validité du bus. Les deux filés d'attentes sont ensuite explorées simultanément pour effectuer les calculs.

A titre d'illustration la figure 5 schématise la configuration nécessaire au processus de la figure 3. Quatre bus banalisés seulement ont été représentés. Ces mêmes modules permettent quantités de configurations allant du terminal classique de la figure 1 au synthétiseur évolué comportant des possibilités d'interaction en temps réel sur les couleurs, sur l'éclairage ou incluant l'élimination des parties cachées à travers l'algorithme du Z-buffer.

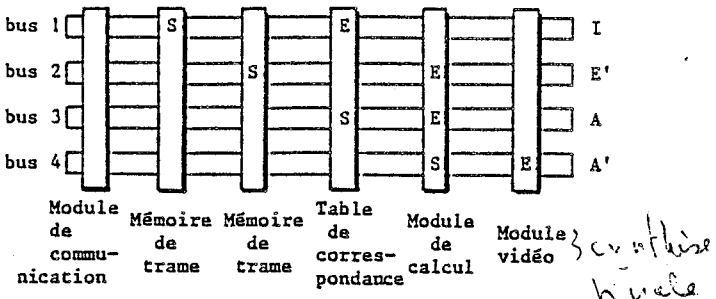


Figure 6

5. CONCLUSION

Le prototype en cours de réalisation est un système expérimental destiné à valider la faisabilité des concepts proposés. Les modules disponibles à l'heure actuelle outre les modules vidéo et communication sont : deux mémoires de trames de 512 × 512 × 12 bits avec translation, fenêtrage et possibilité d'invalider une ou plusieurs informations en temps réel ; deux tables de correspondance (4096 × 12 bits) et un module de calcul d'éclairage. D'autres modules déjà réalisés lors d'une étude précédente[Fer 81] [MaF 82] seront adaptés à cette nouvelle architecture. Il s'agit d'un module de calcul de textures et d'un module permettant le calcul des coefficients de réflexion spéculaire et diffuse en fonction du vecteur normal pour chacun des pixels.

Muni de ces modules ce système permettra de répondre à un grand nombre d'applications allant du graphique bas de gamme jusqu'à la synthèse en temps réel d'images réalistes. L'adaptation à des applications très spécifiques pourra également être réalisée par la simple adjonction d'un module de calcul dédié à cet usage.

Nous pensons que ce type d'architecture peut conduire dans un avenir proche à une unification des synthétiseurs d'images autour d'un bus standard tel que celui qui est proposé dans ces pages. L'utilisateur pourrait alors construire, de manière modulaire, le synthétiseur répondant à ses besoins propres à l'aide des "cartes modules" proposées par différents constructeurs.

REFERENCES

Fer 81 : F. Nunes Ferreira
Conception et réalisation d'un système interactif pour la synthèse d'images réalistes : HELIOS.
Thèse de Docteur-Ingénieur, Grenoble, 9-81.

MaF 82 : F. Martinez, F. Nunes Ferreira
HELIOS : Terminal vidéo interactif pour la synthèse d'images réalistes.
Le Nouvel Automatisme, N° 30, 5-82.

Mar 82 : F. Martinez
Vers une approche systématique de la synthèse d'image : Aspects logiciel et matériel.
Thèse d'Etat, Grenoble, 11-82.

ANNEXE 2 : QUELQUES EXEMPLES DE STRUCTURES GRAPHIQUES ARBORESCENTES

Two methods for improving the efficiency of ray casting in solid modelling

Willem F Bronsvoort, Jarke J van Wijk* and Frederik W Jansen*

In solid modelling based on constructive solid geometry and primitive instancing, ray casting is a very suitable technique for visualization of models on a raster display. In its present form, it is, however, too inefficient for interactive use.

Two methods for improving the efficiency are given here. The first uses scan-line interval enclosures instead of box enclosures, and also bypasses non-contributing nodes during each traversal of the CSG (constructive solid geometry) tree. The second refines the image step by step by subdivision, thereby avoiding explicit computation of the intensities of many pixels of the image. The second method reduces computing time more than the first, but has the disadvantage that slivers may occasionally be lost from the image.

solid modelling, ray casting, efficiency, enclosures, subdivision

In CADCAM systems, the completeness and accessibility of object-geometry information is of central importance in integrating the design, simulation and manufacturing phases. Traditional representation schemes such as the wireframe are inadequate for many CADCAM operations, since their representations of 3D objects are inherently ambiguous, and consequently important geometric properties cannot be calculated.

During the last few years, solid modelling systems have been introduced[1]. These provide unambiguous and informationally complete representations of rigid solid objects. The most important representation schemes currently in use in solid modellers are[2]:

- Primitive-instancing schemes: families of basic objects, eg blocks, spheres, cylinders, cones and tori are provided, and an object is represented by its family name and parameters specifying its dimensions.
- Spatial enumeration schemes: an object is represented by a list of fixed-sized cubes occupied by the object; a reduction in storage requirements and computing time results from octree-representation schemes, in which an object is represented by a hierarchally ordered collection of variably-sized cubes[3].
- Sweep-operation schemes: an object is represented by a contour and a trajectory along which to sweep the contour; translational sweep and rotational sweep are the most common.
- Boundary-representation schemes: an object is represented by dividing its boundary into a number of faces, and representing each face by its edges and vertices.
- Constructive-solid-geometry (CSG) schemes: an object is modelled by transforming primitive objects and combining them by applying the union, intersection and difference operators to the sets of points occupied by the transformed bodies in a 3D coordinate system; composite objects are represented by the transformations and operations to be applied to the constituent primitive objects, which in turn may be represented in any of the previous schemes.

A CSG scheme combined with a primitive-instancing scheme to represent primitive objects is a very elegant and efficient combination for representing solid objects. A composite object is internally represented as a binary tree. At each leaf node is a primitive object; at each intermediate node is the composite object formed by applying the operator at the node to the sub-objects defined by the left and right branches of the node, and at the root node is the complete object. An example of a composite object and the associated binary tree is shown in Figure 1.

Ray casting, or ray tracing as it is often called, is a general technique for rendering shaded images on a raster display[4]. If the image on the screen is envisaged as being produced by a scene behind the screen viewed from an eyepoint in front of the screen, then for every pixel of the image a ray is drawn from the eyepoint through the pixel and cast onto the object (see Figure 2). If only diffuse reflection is taken into account, the first surface in the scene intersected by the ray is the visible one. The shading intensity of the pixel is then determined by the colour of this surface, and the angle between the surface normal at the intersection point and the direction of the light falling onto the surface.

A CSG scheme combined with a primitive-instancing scheme is well adapted to this operation (see Roth[5], the impetus for the work reported here). The main reason for this is the binary-tree structure of the representation. For each ray, the tree is traversed in postorder using recursion. Leaf nodes are visited by determining the parts of the ray — if any — lying inside the corresponding primitive, a relatively simple operation because of the simple structure of the primitives. Composite nodes are visited by combining the parts of the ray lying inside its right and left sub-objects in accordance with the operator at the node. After the root node has been visited, the first point of the remaining parts

Department of Mathematics and Informatics, Delft University of Technology, Julianalaan 132, 2628 BL Delft, The Netherlands

*Department of Industrial Design, Delft University of Technology, Oude Delft 39a, 2611 BB Delft, The Netherlands

 0010–4485/84/010051–05 $03.00

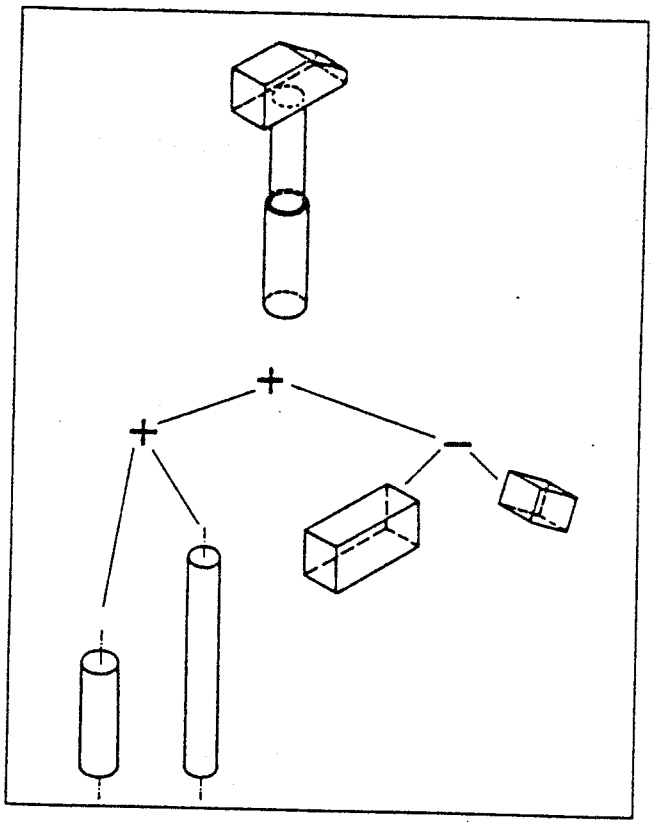


Figure 1. A composite object and the associated binary tree

is the first intersection point of the ray with the object.

At Delft University of Technology we have developed a solid modeller, RAYMO, based on CSG and primitive-instancing representations, which uses ray casting for visualization of models on a raster display. We have concentrated on visualization, because the modeller is primarily for use in industrial design. It is written in Pascal, and runs on a PDP11/44 under RSX-11M. The methods described in this paper have been implemented and tested in RAYMO.

For a solid modeller to be interactive, response times, even for requests for complicated actions, should not exceed a few minutes. On that criterion, ray casting in its present form would have to be combined with some other visualization technique to produce a viable interactive system; response times with our modeller for visualization of moderately complicated objects are of the order of several minutes. However, its advantages over alternative methods in image quality and its conceptual simplicity make it worthwhile to examine ways of improving its efficiency. We present the results of our efforts in the next paragraphs, but first describe briefly an already-reported method for improving the efficiency, that is the starting point for our first method.

The ray-casting procedure just described is straightforward but very inefficient. Because it is called recursively $2n-1$ times for every pixel on the screen, where n is the number of primitive objects, the total number of calls of the procedure may be of the order of millions. This number can be reduced by computing a minimum box enclosure around each primitive object, from which minimum box enclosures around each composite node in the CSG tree are constructed, the box enclosure at each node being determined by the box enclosures around its two sub-objects, and the operator at the node. Then in the tree traversal to

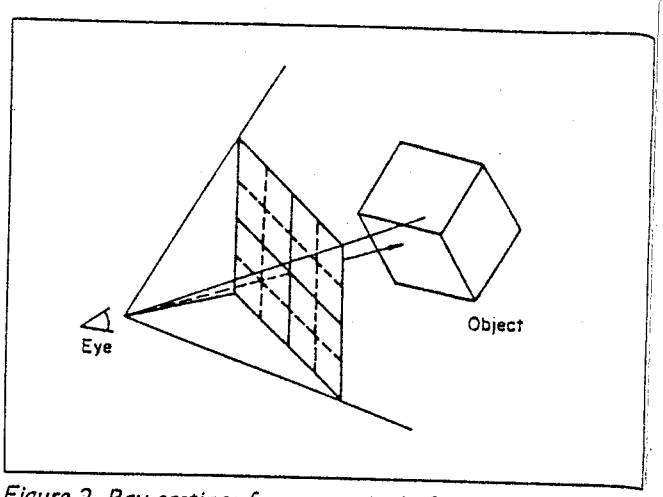


Figure 2. Ray casting: for every pixel of the image a ray is cast from the eyepoint through the pixel onto the object

determine the intersections of a particular ray with the object, tests are carried out at the nodes in preorder to see if the corresponding pixel lies inside the projection on the screen of the corresponding box enclosure. If it does not, the subtrees of the node are not traversed. In this way, many recursive calls of the ray-casting procedure and also many attempted computations of intersection points with primitive objects can be eliminated. See Roth[5] for an elaboration of this technique.

INTERVAL ENCLOSURES AND THE ACTIVE TREE

Our first method arises from noting that if two sub-objects are combined by the union operator, the box enclosure of the composite object is too coarse an indication of the volume occupied by that object. In our modification, the 'ray intersects box' test is reduced to a 'point in scan-line interval' test. The interval enclosure may vary from scan-line to scan-line. Bypassing non-contributing nodes during each traversal of the CSG tree then becomes possible, thereby further reducing the number of recursive calls of the ray-casting procedure. The method resembles the use of an active list in scan-line algorithms[6], using an active tree instead.

A defect of box enclosures

Although box enclosures are simple and effective, a box enclosing an object composed of two sub-objects combined by the union operator often contains large volumes outside both sub-objects and therefore outside the composite object. As a 2D example, consider the union of two sub-objects with box enclosures A and B shown in Figure 3. All points inside the boxes C and D are outside the composite object, but included in the box around this object $(A+B+C+D)$.

Boxes that are too coarse can arise close to leaf nodes and then have a cumulative effect at union operations at all higher-level nodes in the CSG tree. This may cause a lot of rays that are outside all boxes at the leaf nodes to be classified as inside boxes at composite nodes, and thus a lot of superfluous calls of the ray-casting procedure.

The use of intervals instead of boxes

If the pixel values of the screen are computed in scan-line order, we can use intervals enclosing the parts of the primitive objects on the current scan-line. Interval

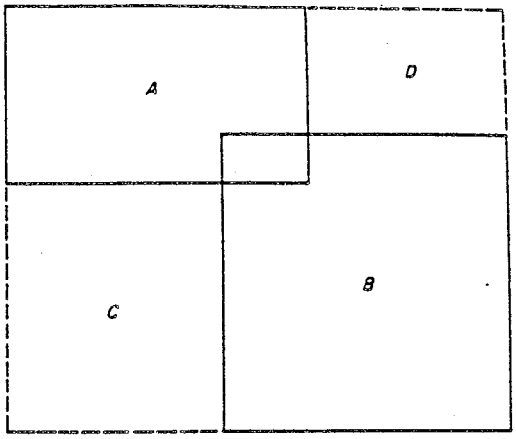


Figure 3. The box enclosure resulting from applying the union operator to the box enclosures A and B

enclosures for the composite objects at higher levels in the tree can be computed in the same way as for box enclosures. The 'ray intersects box' test at each node in the tree traversal is reduced to a 'point in interval' test. The lengths of the interval enclosures are mostly less than, but in the worst case equal to, the lengths of the original box enclosures, thus increasing the chance that a ray outside some sub-object will be classified as such earlier in the tree traversal, thereby decreasing the number of recursive calls of the ray-casting procedure.

The interval enclosures may remain the same for several successive scan-lines, so it would be inefficient to recompute them for every scan-line. To avoid this, boxes enclosing the primitive objects are computed first. All minimum and maximum vertical Y-values of these boxes are collected in an array and sorted. The minimum and maximum horizontal X-values of each box, bounding the interval enclosure of the primitive object, are stored at the corresponding leaf node. At the transition from one scan-line to the next, the array is checked to see if a minimum or maximum Y-value is passed. If so, the interval enclosures of the composite nodes are recomputed, using the interval enclosures of the primitive objects.

The active CSG tree

If interval enclosures are recomputed at the appropriate scan-lines, another improvement becomes possible: non-contributing nodes can be bypassed during each CSG tree traversal. Consider a node A having subnodes B and C, with B having subnodes D and E (see Figure 4).

If the operator at node B is the union operator, and node D has a non-empty interval enclosure and node E an empty interval enclosure, traversal of nodes B and E is needless, because the 'point in interval' test at B is identical to the test at D, and the test at E gives a negative result by definition. The superfluous calls of the ray-casting procedure can be avoided by connecting node A directly to node D, thus bypassing the non-contributing nodes B and E during a tree traversal. In the same way, nodes B and D can be bypassed if D has an empty interval enclosure.

If the operator at node B is the difference operator, and node D has a non-empty interval enclosure and node E an empty interval enclosure, traversing nodes B and E is again needless, and node A can again be connected directly to node D. If, however, node D has an empty interval enclosure, node B will have an empty interval enclosure too, and nodes B, D and E can all be bypassed.

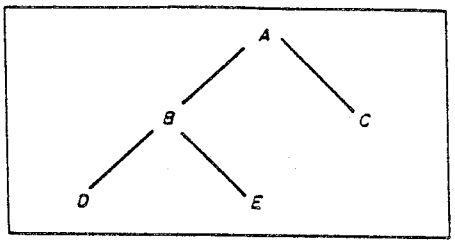


Figure 4. Part of a CSG tree

If the operator at node B is the intersection operator, and node D or node E has an empty interval enclosure, node B will have an empty interval enclosure too, and nodes B, D and E can again all be bypassed.

The active CSG tree, containing only the currently contributing nodes of the tree, is recomputed simultaneously with the interval enclosures. To implement the active tree, all composite nodes contain, besides pointers to its left and right subtrees in the complete tree, pointers to its active subtrees. Searching for non-contributing nodes takes place at all levels in the tree, so that the active tree can be considerably smaller than the complete tree. This again decreases the number of recursive calls of the ray-casting procedure.

STEPWISE REFINEMENT OF THE IMAGE BY SUBDIVISION

Our second method is based on the notion of coherence in the intensities of neighbouring pixels in the image. The stepwise refinement algorithm starts by computing the image with a low resolution, eg with squares of 8 x 8 pixels, and subsequently refines this image to one with a higher resolution by subdivision of these squares. This is done by comparing the intensities of neighbouring subsquares and recomputing intensities only if these are significantly different. The method is based on an idea of Roth[5] of sparsely sampling an image and locating edges via binary searches, and on an idea of Whitted[4] of anti-aliasing an image by subdividing a pixel whose corner intensities are not nearly equal. A disadvantage of this algorithm and ways in which the user can control its behaviour are also discussed.

The stepwise refinement algorithm

Both the box-enclosure and the interval-enclosure algorithms compute the intensity of all pixels in the image taking no account of the intensities of neighbouring pixels. However, neighbouring pixels often have the same, or approximately the same intensity, in particular if they are on one face of an object. This notion of coherence is utilized in the stepwise refinement algorithm to speed-up the computation of an image.

The idea is to start off by computing the image with a low, user-specified resolution. The screen is divided into squares with sides of length 2^n pixels, eg n=3. The intensity of the pixel at the lower-left corner of a square is computed explicitly, and all other pixels inside the square get the same intensity. The resulting image is a coarse approximation of the final image.

Next, all squares are subdivided into 4 equal subsquares, as shown in Figure 5. The intensities of the 4 subsquares $X1$, $X2$, $X3$ and $X4$ are now computed as follows:

- The intensity of the lower-left subsquare *X1* remains the same.
- If one of the current intensities of the squares denoted by *A* differs more than a threshold value from the intensity of *X1*, the intensity of the lower-left pixel of *X2* is computed and all other pixels inside *X2* get the same intensity.
- If one of the current intensities of the squares denoted by *B* differs more than the threshold value from the intensity of *X1*, the intensity of *X3* is recomputed in the same way.
- If one of the current intensities of the squares denoted by *C* differs more than the threshold value from the intensity of *X1*, the intensity of *X4* is recomputed in the same way.

This computation proceeds from the lower rows with squares to the higher ones, and from left to right on a row.

The subdivision process is continued further until the final, user-specified resolution, that may have squares of more than one pixel, is reached. It is also possible to continue the process to subpixel level, resulting in an anti-aliased image. The image is thus stepwise refined, and the user can see this happening on the display.

It turns out that the number of pixels that has to be computed explicitly, ie with a separate call of the ray-casting procedure, is significantly less than the total number of pixels in the image. The computation is concentrated around those areas of the image with large intensity gradients, eg the edges of the object, while the smooth-shaded areas, eg the faces of the object, require less refinement.

A disadvantage of the algorithm

A disadvantage of the stepwise refinement algorithm is that slivers may occasionally be lost from the final image, because the intensity of many pixels is based on the intensities of neighbouring pixels and is not computed explicitly, which may result in an incorrect value.

Two factors that may influence the final result are the initial resolution and the threshold value. The user of the system can control its behaviour by specifying the initial resolution and determining the threshold value at each successive step of the algorithm.

In its simplest form, the value of the threshold is a constant. However, much CPU time can be saved by increasing the value at higher resolutions. Only those areas of the image where it is absolutely necessary are then further refined, while those areas that are already acceptably shaded are left unaltered. In particular at the anti-aliasing step this approach is very effective.

In many circumstances, the user may be satisfied with a fast-generated image of low resolution with possibly some minor inadequacies. The stepwise refinement algorithm offers a suitable previewing mechanism for this purpose[7].

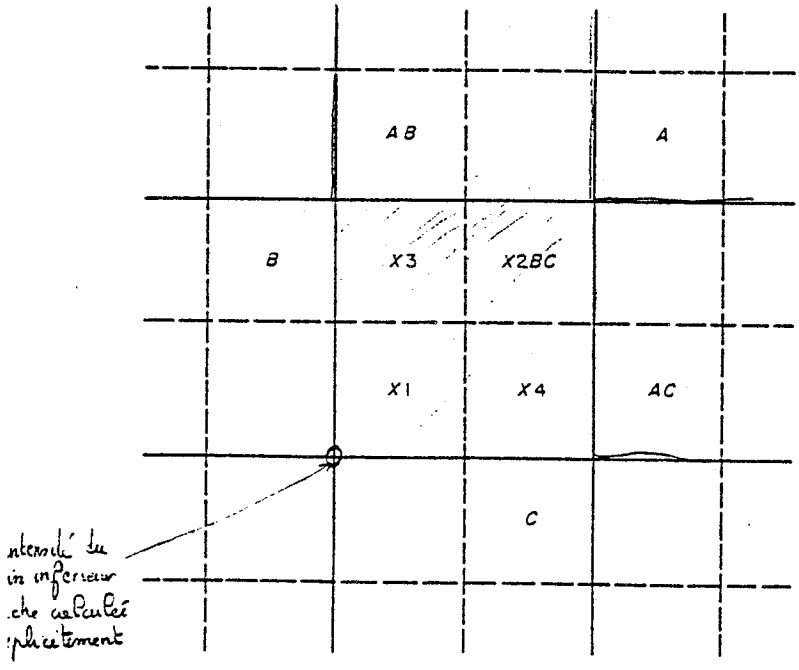


Figure 5. Subdivision of squares into 4 equal subsquares

RESULTS AND DISCUSSION

Some measurements are presented, on the basis of which assessment can be made of the improvements in efficiency brought about by both methods separately, and in combination. Finally, some observations on further research are made.

An evaluation of the two methods

A comparison was made of three implementations, one based on box enclosures, the second on interval enclosures and an active tree, and the third on stepwise refinement combined with box enclosures. The results for the visualization of several objects using the last two methods were compared to the results using the first method. Three of these objects are shown in Figure 6. The criteria for comparison were the number of calls of the ray-casting procedure and the CPU time which each required. The improvements due to the two methods are, as expected, heavily dependent on the geometry of the object being modelled and visualized.

The use of interval enclosures and an active tree considerably reduces the number of recursive calls of the ray-casting procedure (about 25—45 per cent), because at each scan-line the interval enclosures and the structure of the tree are optimized to restrict the traversal of the CSG tree. The reduction in CPU time is not commensurate (about 5—15 per cent), partly because of the time needed to recompute the interval enclosures and the active tree at the appropriate scan-lines, and partly because the computation of the intersection points with the primitive objects is dominant. The effect of the method is most significant for objects composed of many primitive objects with few overlaps. The method is very easy to implement.

The stepwise refinement method, starting off with squares of 8 x 8 pixels and ending with squares of 1 pixel, also considerably reduces the number of procedure calls (about 45—60 per cent), and reduces the CPU time much more than the first method (also about 45—60 per cent), because the coherence in intensity of neighbouring pixels in the image makes explicit computation of the intensity of many pixels unnecessary. If at square size of 2 x 2 pixels the threshold value is raised from the starting value of 0 per cent of the maximum difference in intensity values to 5 per cent, and at square size of 1 pixel to 15 per cent, the effect is even larger (a reduction of about 65—80 per cent in number of procedure calls and also of about 65—80 per cent in CPU time). Large areas of the image receive no further refinement, without noticeable loss in the quality of the image. A disadvantage of the method is that slivers may occasionally be lost from the image. Its effect is most

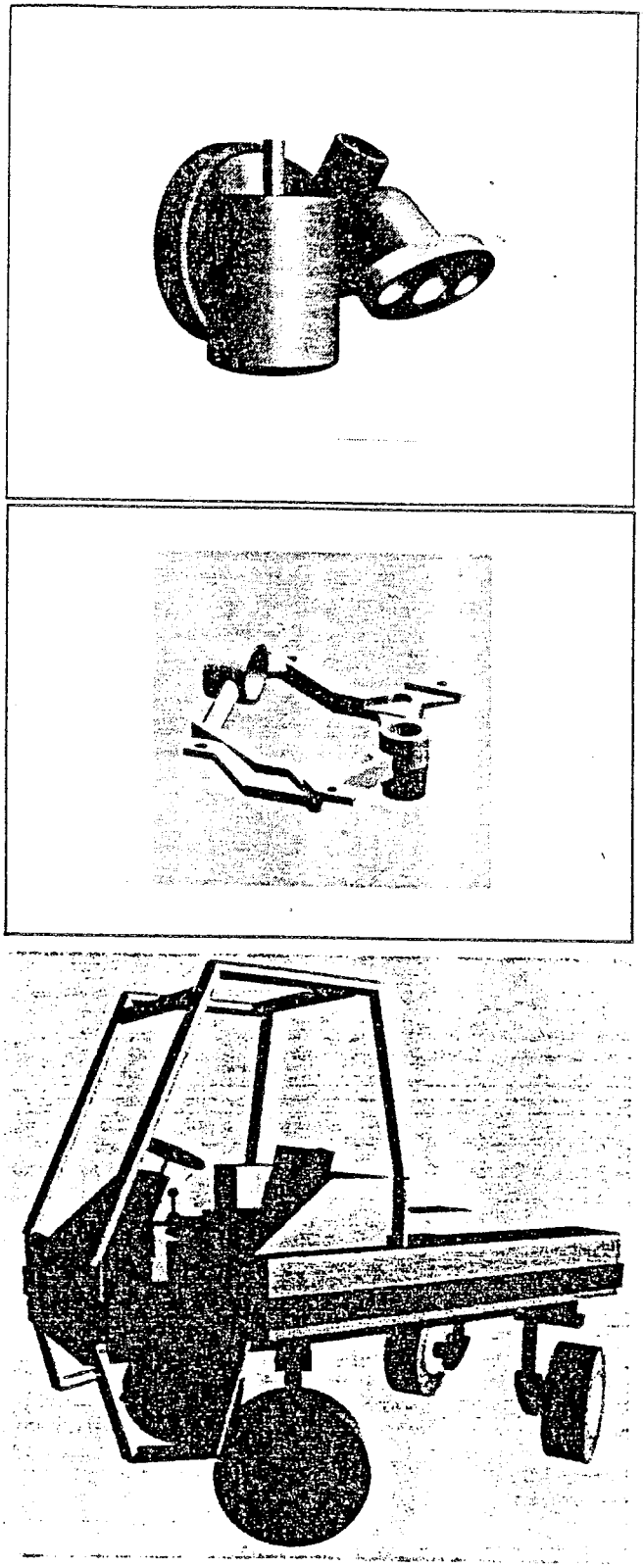


Figure 6. A carburettor, a casting and a forester's vehicle modelled and visualized with RAYMO, and used for comparison of the methods reported

significant for objects with nearly flat surfaces, few edges and few fine details. This method is also very easy to implement.

Combining the two methods

The methods of interval enclosures together with an active tree and of stepwise refinement can be combined in one algorithm, which makes the computation even more efficient. If the threshold value is increased at the higher resolutions, the total reduction in number of procedure calls is about 80–90 per cent and in CPU time about 70–85 per cent.

Discussion

Although the results of the two methods presented show reasonable improvements in computing times for images of some complexity, the times still remain too long for interactive use.

With the method of interval enclosures and an active tree, the necessary traversal of the tree is considerably reduced. This probably eliminates the usefulness of automatically reorganizing the complete CSG tree, as suggested[5].

Atherton[8] has recently published a scan-line algorithm for ray tracing in solid modelling. The improvements in efficiency he reports are remarkable. However, objects in his modeller have to be described in terms of polygons, a restriction to which our modeller is not subject.

We have tried to improve the efficiency in two ways. The first relies on environmental coherence; it computes the intensity of every pixel separately, but restricts the traversal of the CSG tree more efficiently than the box-enclosure method. The second relies on image coherence; it samples the image for some pixels and predicts the intensities of the remaining pixels on the basis of this sampling. The results indicate that additional inherent improvements are most likely to be found through improved techniques of sampling and predicting.

ACKNOWLEDGEMENTS

We would like to thank D J McConalogue for many valuable comments on earlier versions of this paper. The work of J J van Wijk is supported by a grant from the Delfts Hogeschoolfonds.

REFERENCES

1 Voelcker, H B and Requicha, A A G (eds) Special issue on solid modeling *IEEE Comput. Graphics Appl.* Vol 2 No 2 (March 1982)

2 Requicha, A A G 'Representations of rigid solids: theory, methods and systems' *Comput. Surveys* Vol 12 No 4 (December 1980) pp 437–464

3 Meagher, D 'Geometric modeling using octree encoding' *Comput. Graphics Image Proc.* Vol 19 No 2 (June 1982) pp 129–147

4 Whitted, T 'An improved illumination model for shaded display' *Commun. ACM* Vol 23 No 6 (June 1980) pp 343–349

5 Roth, S D 'Ray casting for modeling solids' *Comput. Graphics Image Proc.* Vol 18 No 2 (February 1982) pp 109–144

6 Foley, J D and Van Dam, A *Fundamentals of interactive computer graphics* Addison-Wesley, MA, USA (1982)

7 Jansen, F W and Van Wijk, J J 'Fast previewing techniques in raster graphics' *Proc. Eurographics '83* North-Holland, Amsterdam, The Netherlands (1983)

8 Atherton, P R 'A scan-line hidden surface removal procedure for constructive solid geometry' *SIGGRAPH '83 Conf. Proc.* ACM, USA (1983)

Graphics and
Image Processing

Hierarchical Geometric Models for Visible Surface Algorithms

James H. Clark
University of California at Santa Cruz

The geometric structure inherent in the definition of the shapes of three-dimensional objects and environments is used not just to define their relative motion and placement, but also to assist in solving many other problems of systems for producing pictures by computer. By using an extension of traditional structure information, or a geometric hierarchy, five significant improvements to current techniques are possible. First, the range of complexity of an environment is greatly increased while the visible complexity of any given scene is kept within a fixed upper limit. Second, a meaningful way is provided to vary the amount of detail presented in a scene. Third, "clipping" becomes a very fast logarithmic search for the resolvable parts of the environment within the field of view. Fourth, frame to frame coherence and clipping define a graphical "working set," or fraction of the total structure that should be present in primary store for immediate access by the visible surface algorithm. Finally, the geometric structure suggests a recursive descent, visible surface algorithm in which the computation time potentially grows linearly with the visible complexity of the scene.

Key Words and Phrases: visible surface algorithms, hidden surface algorithms, hierarchical data structures, geometric models
CR Categories: 5.31, 8.2

A version of this paper was presented at SIGGRAPH 76: The Third Annual Conference on Computer Graphics, Interactive Techniques, and Image Processing, The Wharton School, University of Pennsylvania, July 14–16, 1976.

Author's address: Information Sciences, University of California, Santa Cruz, CA 95064.

Communications of the ACM
October 1976
Volume 19
Number 10

1. Introduction

1.1 Background

Early research in computer graphics was concerned with the organization and presentation of graphical information in the form of real-time line drawings on a CRT. Many of the concepts of structuring graphical information were developed by Sutherland in Sketchpad [19], and the line-drawing graphical displays that resulted from his early research remain the most widely used today. With the development of integrated circuit technology, research interests shifted to producing very realistic, shaded, color pictures of the visible parts of complex three-dimensional objects. Because of the desire to utilize television technology, the algorithms for producing these pictures generated output for a raster CRT. The pioneering works in this area were by Schumacker et al. [18] and Wylie et al. [23].

Computer produced pictures now provide one of the most direct and useful ways of communicating with the computer. The ability to produce shaded pictures that illustrate mathematical functions and physical properties of mathematical models is of incontestable value in both research and education. With the development of computer controlled simulators, a real-time computer displayed environment is now used to train pilots of aircraft [11, 16], spacecraft [9] and ocean vessels [2]. Other significant uses of computer pictures include computer aided design [4], modeling of chemical structures [22], and computer animation [7, 12]. With this increased value of computer generated pictures, comes an increasing need to devise efficient algorithms that improve the realism and enhance the descriptive power of these pictures.

1.2 Motivation for New Research

The underlying motivation for new research on computer produced pictures is to either enhance the realism of the pictures or improve the performance of the algorithms that generate them. Most recent research has addressed a combination of these issues.

There are three basic approaches to improving picture quality. The first is to devise clever ways to add information value to a scene without significantly increasing the total amount of information in the database for the scene, for example, without increasing the number of polygons used in representing the objects. Approaches of this type usually make subtle changes to the visible surface and shading algorithms that result in greatly improved pictures. Examples are the improvements to shading algorithms devised by H. Gouraud [10] and Bui-Tuong Phong [15].

The second approach is to employ more refined mathematical models for the objects being rendered and to devise algorithms that can find the visible surfaces using these models. The goal of these methods is to model smooth surfaces with surface patches, such as Coons patches [5] or B-splines [4, 17], rather than with clusters of polygons, and still not increase the size of the database. Catmull's [3] ingenious algorithm is an example of this approach. The benefit of these methods is that an arbitrarily refined description of the model is present, thus allowing much better renditions of contours and shading. The disadvantage is that because of nonlinear mathematics, the algorithms are less efficient than polygon-based algorithms.

The third approach is to increase the information in the database and employ more structured methods for handling the increased information. The motivation for this approach is that the information value of a scene grows in proportion to the amount of information in the database for the scene. Newell's [13] algorithm is an example of this approach.

The structured approach appears to be the most promising of these approaches since it potentially improves both picture quality and algorithm performance. However, there are several problems associated with this approach. First, increased complexity of a scene, or increased information in the database, has less value as the resolution limits of the display are approached. It makes no sense to use 500 polygons in describing an object if it covers only 20 raster units of the display. How do we select only that portion of the data base that has meaning in the context of the resolution of the viewing device? Second, how do we accommodate the increased storage requirements of this additional information? We might, for example, wish to model a human body to the extent that a closeup view of the eye shows the patterns of the iris, yet such a fine description of the entire body will indeed require large amounts of store. Third, how much information must be presented to convey the information content of the scene? In other words, we would like to present the minimal information needed to convey the meaning of what is being viewed. For example, when we view the human body mentioned above from a very large distance, we might need to present only "specks" for the eyes, or perhaps just a "block" for the head, totally eliminating the eyes from consideration. The amount of information "needed" can be the subject of psychological debate, but it is clear that even coarse decisions will yield more manageable scenes than attempting to use all of the available information.

These issues have not previously been addressed in a unified way. The research described here represents an attempt to solve these and related problems.

2. Summary of Existing Algorithms

Visible surface algorithms may be categorized according to whether they employ polygons, parametric surface patches, or procedures as the underlying method of modeling the surfaces they render. The most thoroughly studied types of algorithms use polygons. However, because of the shortcomings of representing

smooth surfaces with faceted clusters of polygons, some research interest has recently been devoted to parametric surface algorithms, which allow higher degrees of continuity than just positional continuity. The algorithms for these different modeling methods will be discussed separately.

2.1 Polygon-Based Algorithms

A highly informative survey of existing polygon-based visible surface algorithms has been written by Sutherland et al. [20]. As they point out, a convenient way to classify these algorithms is according to the order in which they sort the image space polygons that are potentially visible in a scene. The basic difference between the major algorithms is in whether they sort in depth (from the viewpoint) before the vertical-horizontal sort, or vice versa.

Depth-first sort. The most significant algorithms to use this sorting order are due to Schumacker et al. [18] and Newell et al. [14]. Schumacker utilizes this order along with a polygon clustering concept to achieve a coherence from one frame to the next, while Newell utilizes it to render translucent images. By first computing a priority ordering of polygons according to their image space distance from the screen, they are able to establish which polygon *segments* on a given scan line have visibility priority.

Newell uses this information to write those segments with a lesser priority into a scan-line buffer before writing in those with a greater priority. Thus greater priority segments which are from translucent polygons only modify the intensity values in the buffer rather than completely overwriting them. While there is clearly a considerable overhead in writing into the buffer segments that might eventually be obscured, some beautiful pictures have resulted from this work.

Schumacker's goal is to produce real-time picture sequences. Rather than writing the polygon segment information for a scan-line into a buffer according to its priority, a set of priority-ordered hardware registers are simultaneously loaded with the priority-ordered segment information. Then as the scan line is displayed, the register information is counted down and a combinational-logic network selects the appropriate highest priority register according to its lateral displacement on the screen. This approach requires a separate set of registers for each polygon segment that intersects the scan line. Nonetheless, it represents the first real-time solution to the visible surface problem [9].

There are two very significant features to Schumacker's work. First, he makes use of a priori knowledge of the database to compute fixed priorities for clusters of polygons. If the polygons in a group of polygons are not subject to changes in relative placement, they form a *cluster* and may be assigned fixed priorities which work no matter from where the cluster is viewed. Thus part of the priority ordering is fixed with the environment and need not be recomputed each frame. Second, he shows that if the environment is restricted so that the clusters are linearly separable, an intercluster priority can be established that does not change unless the viewpoint crosses one of the separating planes; hence, the priority ordering remains fixed from one frame to the next unless one of the planes is crossed.

This work by Schumacker and coworkers represents the only visible surface algorithm to make use of both structured information (clustering) and frame to frame coherence (relatively constant intercluster priority). These very important concepts will be discussed in more detail later.

Depth-last sort. The algorithms that use this sorting order have been devised by Watkins [21], Bouknight [1], and Wylie et al. [23]. They are referred to as scan-line algorithms and differ only in their use of various image-space coherence properties. All three first perform a vertical bucket (radix) sort of polygon edges according to their uppermost vertices. Then for each scan line, the various polygon segments on that scan line are sorted according to their horizontal displacements. The depth sort is deferred until last under the assumption that the initial two sorts will decrease the number of depth comparisons needed to determine final visibility.

Of the three approaches, Watkins' is the most economical because of its uses of scan-line coherence and a logarithmic depth search. The assumption of scan-line coherence is that in going from one scan line to the next, the number of changed polygon segments is small; hence the horizontal sort may be optimized to take advantage of this. Watkins' is the only other algorithm besides Schumacker's that has been implemented in hardware.

2.2 Parametric Surface Algorithms

Modeling smooth surfaces with collections of polygons leads to problems both in shading the surface and in rendering the contour edges. While there have been a number of very clever improvements to the quality of such pictures without significantly increasing the amount of information used, notably those of Gouraud [10], Phong [15], and Crow [6], the most direct approach is to employ a more refined model, such as parametric surface patches. Such patches can be used to define the surface using no more, and usually even less, information than is required with polygons. Yet they can join together with tangent or even higher continuity, thus eliminating the above problems. The difficulty with this method is that the mathematics is no longer linear; to explicitly solve for such things as the curve of intersection of two bi-cubic patches or of a patch and a clipping plane are very difficult problems.

Catmull [3] solves such problems, but not explicitly. Rather, he does so by employing the discrete character of the image space, a recursive algorithm, and what he calls a Z-buffer. For each patch in the

environment his algorithm asks: does the patch extend over more than a single raster unit? If the answer is yes, the patch is subdivided (by a very fast algorithm for bi-cubic patches) into four patches and the same question is *recursively* asked of these patches. When the answer finally is no, an attempt is made to write the intensity and depth coordinates for the resulting "patch" into a buffer for the raster unit, or *pixel*, in question. The attempt fails if the pixel buffer already has in it a depth coordinate nearer to the observer (with some minor modifications to allow for translucent patches).

A very significant feature of Catmull's algorithm is that, despite the more complex mathematics, it will actually work faster than polygon-based algorithms if the object being rendered occupies a very small area of the screen. Because of the recursive structure of the algorithm, it will "structure" the surface no more finely than the resolution of the display dictates, whereas current polygon-based algorithms keep the same structural description, i.e. the same number of polygons, no matter how much of the screen area is occupied. This notion of structuring will be extended to include polygon-based algorithms in the next section.

2.3 Procedurally Modeled Objects

Newell [13] has recently employed procedural modeling to solve the visible surface problem for complex scenes. According to this approach, objects are modeled using procedures which "know how" to render themselves in terms of their own primitives, which might include activations of other object procedures; such knowledge includes rendering only their visible parts. This is a very general way to represent objects.

Although the underlying philosophy of this approach is very general, in the actual implementation Newell user polygons as the basic primitives for the objects. The object procedures are activated according to a priority ordering so that more distant objects are activated first. Each procedure renders the object it represents by activating the Watkins process, the results of which are written into a frame buffer. The net result is therefore a "hybrid" Watkins/Newell priority algorithm.

The significant point about this algorithm is not the procedural modeling but that it represents another example of structuring to simplify the total sorting problem, namely that the geometric primitives of one object need be compared with those of another only when the objects overlap.

3. Hierarchical Approach

It was indicated in the previous section that, aside from uses of image-space coherence to reduce the amount of sorting required, the most fruitful gains in visible surface algorithm research have resulted from structuring the environments being rendered. However, the structures employed take a diverse variety of forms, from Catmull's implicit structuring of surface patches to Newell's procedural objects. What is needed is a single, unified, structural approach that embodies all of the ideas from these algorithms. Before presenting one such approach it is instructive to consider two ways in which structure has been utilized to prepare objects for visible surface processing.

3.1 Existing Uses of Structure

Defining relative placement. The benefits of a position or motion structure have been realized for some time. Sutherland used such concepts in two dimensions in Sketchpad, and a number of graphics hardware companies incorporate transformation hardware in their display devices to accommodate structural descriptions. Most of the visible surface algorithms presented used a position or motion structure to describe positions and orientations of objects relative to each other. However, all but the few mentioned in Section 2 disregard the structure at the visible surface algorithm level. That is, all polygons of the objects are transformed into a common screen coordinate system in which the visible surface algorithm works.

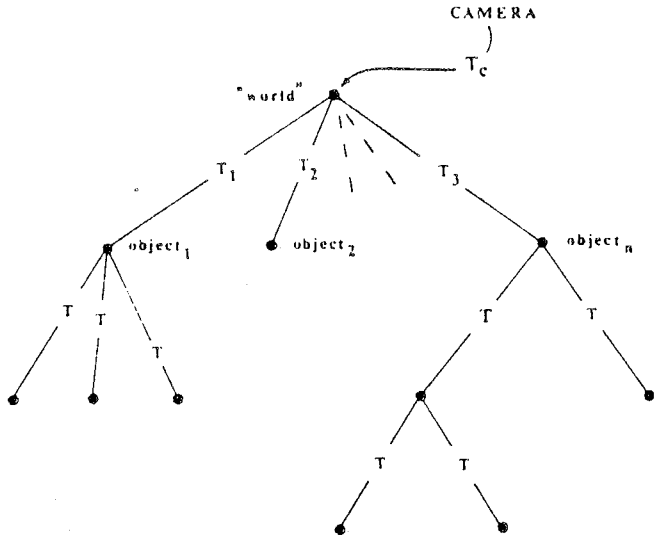
An example of such a structure is shown in Figure 1. Each node in the hierarchy represents a set of geometric primitives (e.g. polygons) defining the node and the arc leading to the node represents a transformation defining the orientation and placement of the node relative to its "parent." Because each node has its own unique transformation defining it, it may represent one of many "instances" of the same primitive description, or data set. This is a very convenient and general way to define and place objects.

Decreasing clipping time. When simulating a camera in a computer-generated environment, some parts of the environment must be "clipped" to the field of view of the simulated camera. This can be done either by transforming all of the geometric primitives of each object into the camera, or screen, coordinate system and clipping each of them separately or by first clipping some bounding volume of the object to see if it intersects the boundaries of the field of view. If it does not, then the parts of the object lie either totally within or totally outside of the field of view and thus need not be separately clipped. This utilization of the above mentioned position hierarchy is implicitly assumed, although this author does not know if the authors of the various algorithms actually made such use of it.

3.2 New Uses of Structure

Varying environment detail. By choosing to represent an object with a certain amount of detail, one fixes the minimum distance from which the object may be displayed with a realistic rendering. For example, a dodecahedron looks like a sphere from a sufficiently large distance and thus can be used to model it so long

Fig. 1. The traditional motion structure used to position objects relative to the "world" and subobjects relative to objects. Each arc in the graph represents a transformation.



as it is viewed from that or a greater distance. However, if it must ever be viewed more closely, it will look like a dodecahedron. One solution to this is simply to define it with the most detail that will ever be necessary. However, then it might have far more detail than is needed to represent it at large distances, and in a complex environment with many such objects, there would be too many polygons (or other geometric primitives) for the visible surface algorithms to efficiently handle.

As mentioned in Section 2, the solution to this problem has been to define objects relatively coarsely and employ clever algorithms that smooth appropriate contours or improve shading to make the object look more realistic at close observation. The difficulty with these approaches is that at best the range of viewing depth is only slightly improved, and the problem of too much detail at large distances usually remains. Although these approaches have yielded results of unquestionable value, it seems evident that multiple levels of description must be used to adequately represent complex environments.[1]

How does one represent these multiple levels of description? A solution is to define "objects" in a hierarchy like that of Figure 2. The entire environment is itself an "object" and is represented as a rooted tree. ("Object" is a generic term for the things represented by nodes of the tree. This generic term will be used for the remainder of this paper.) There are two types of arcs in the tree, those that represent transformations as before and those that represent pointers to more detailed structure (the identity transformation). Each nonterminal node represents a "sufficient" description of the "object" if it covers no more than some small area of the display; the arcs leading from the node point to more detailed "objects" which collectively define a more detailed version of the original object if its description is insufficient because it covers a larger area of the screen. The terminal nodes of the tree represent either polygons or surface patches (or other primitives) according to whether they are primitive elements of a faceted or a smooth object.

As an example of such a description, consider a model of the human body. When viewed at a very large distance, for example when the body covers only 3 or 4 display raster units, it is sufficient to model the body with a single rectangular polyhedron with appropriate color. Therefore the uppermost node, or "object," for this body represents this simple description. If the body is viewed from a closer distance—for example, if its topmost node's description covers 16 raster units—then this topmost description is no longer sufficient, and the next level of more refined description is needed. At this next level the body is now perhaps described as a collection of rectangular polyhedra appropriately attached to each other, for example using one polyhedron for each of the arms and legs, the head and the torso. Then so long as each of these "objects" covers only a few raster units of the display, their description is "sufficient." When the viewing distance decreases such that any of them covers a critical maximum area of the display, its more detailed subobjects are used to replace its description. This process is carried out to whatever maximum level of detail will be needed. For example, a terminal level of description of the fingertip might be several surface patches (which could be implicitly structured even more finely using Catmull's algorithm).

The body described is just one "object" of an environment, or larger hierarchy. There might be many such bodies, or other objects. The significant point, however, is that in a complex environment, the amount of information presented about the various objects in the environment varies according to the fraction of the field of view occupied by these objects.

It is worth noting again that Catmull's algorithm, described in the previous section, implicitly built such a structure. His algorithm used this structure in such a way that, despite the more complex mathematics of surface patches, it outperforms polygon-based algorithms if the surface occupies a small area of the screen. Thus it seems that such a structure should lead to improvements in polygon-based algorithms as well.

Clipping: a truncated logarithmic search. The choice of this structural representation poses another problem. How does one select only that portion of a potentially very large hierarchy that is meaningful in the context of the viewpoint and the resolution of the viewing device? In other words, clipping in a broader sense must mean selecting not only that part of the environment within the field of view (the usual meaning) but also just the *resolvable* part. This implies finding the visible nodes of

[1] Actually, Evans and Sutherland made use of a three-level description of the New York skyline in its Maritime simulation, but in an ad hoc way [2].

the tree, as shown in Figure 2. The contour shown in the figure represents a possible set of objects that are within the field of view and are both not too large and not too small for the screen areas they occupy.

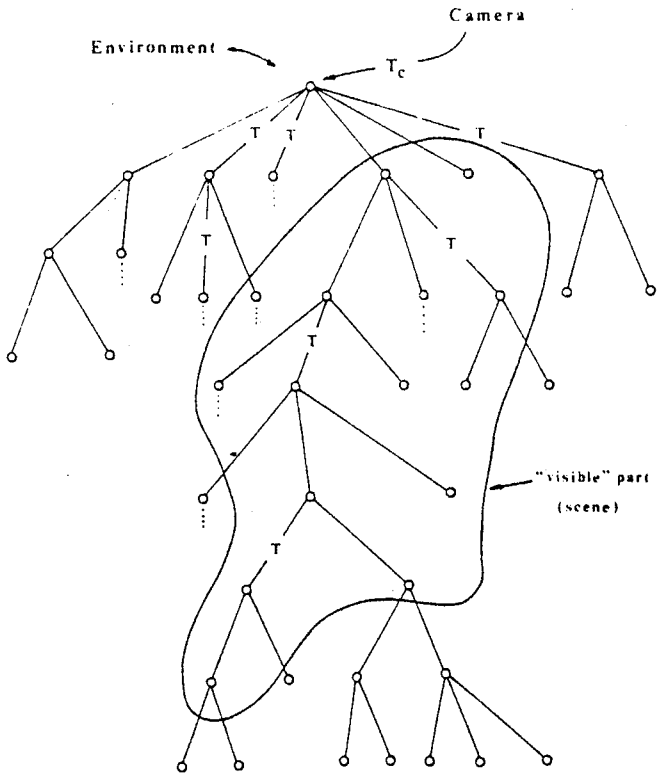
In order to efficiently perform this clipping operation some minimal description of object sizes must be available. For example, a bounding rectangular box or a bounding sphere would be sufficient information to test whether an object is totally within or totally outside of the field of view. The minimum necessary information is the center and radius of a bounding sphere.

This general structure therefore suggests a very fast clipping algorithm which recursively descends the tree, transforming (if necessary) this minimal information into perspective viewing coordinates and testing both the area occupied by the bounding sphere and its intersection with the boundaries of the field of view. The criterion for descending a level is the area test, while the criterion for inclusion/rejection is the field of view boundary test. Only after either the area test terminates the descent or the terminal level of representation is reached is it necessary to actually transform and possibly clip the polygons or surface patches represented by the node. Clipping therefore resembles a logarithmic search that is truncated by the area (resolvability) test.

This relatively simple mechanism for varying the detail in a scene suggests several other interesting possibilities. Since the center of attention of a scene is often its geometric center, one might effectively render the scene with a center-weighting of detail. In other words, the maximum area an object is allowed to cover before splitting it into its subobjects becomes larger towards the periphery of the field of view. This is somewhat analogous to the center-weighted metering systems of some cameras. Likewise, since moving objects are less resolved by both the human eye (because of saccadic suppression) and a camera (because of blurring), one can render them with an amount of detail that varies inversely with their speeds. Indeed, an entire scene might be rendered with less detail if the camera is moving. Thus "clipping" can be extended to include these concepts as well.

Graphical working set. Since the problems addressed by this model are those associated with producing pictures and picture sequences of very complex environments, the excessive storage needed for the geometric description of these environments must somehow be accommodated. Denning's "working set" model for program behavior provides a useful analogy [8]. According to this model, a computer program that makes excessive demands on immediate-access store is structured or segmented, and its storage demands are managed in such a way that only those segments most recently in use are actually kept in immediate-access store. The remaining potentially large number of segments are kept on a slower, secondary store, such as a disk. The "working set" is that set of segments

Fig. 2. A very deep hierarchy that structures the environment much more than the traditional motion structure. Arcs in this graph represent either transformations or pointers to more refined definitions of the node. The visible part contour represents a possible result of clipping.



available for immediate access, and is usually defined by a time average of past program reference patterns. Reference to an unavailable segment causes that segment to become part of the working set, and segments not accessed after some period of time are deleted from the working set.

This working set model coupled with the broader sense of clipping mentioned above suggests a suitable way to accomplish a particular type of frame coherence. The working set in this context is that set of objects in the hierarchy that are "near" to the field of view, inside it, or "near" to the resolution of the image space. Only if an object is a member of this set is its description kept in immediate-access store. The set membership will change slowly since the differences between one scene and the next are usually small. Those cases in which the differences are large due to fast camera (or object) motion are easily accommodated by rendering the scene (or object) with less detail, as mentioned above. Moreover, the minimal description of node size needed for clipping suffices as the graphical analog of the segment table used in the computer program context. That is, this minimal clipping description must always be available in immediate-access store to facilitate determining the working set. This working set model therefore seems particularly well suited to the graphics context.

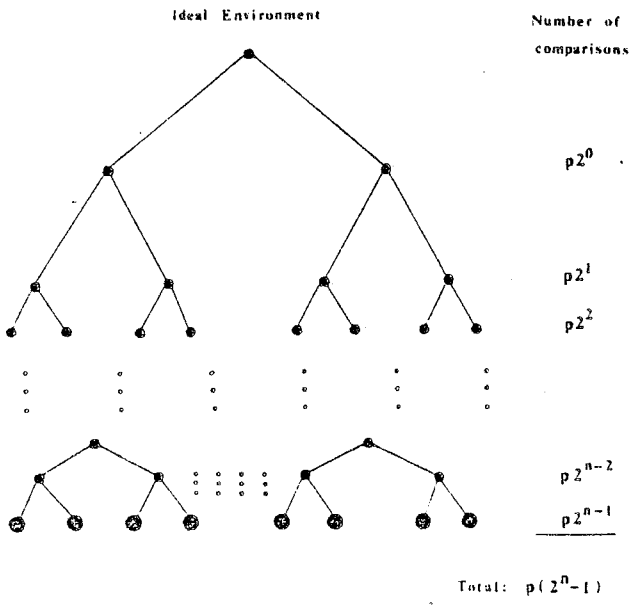
Improving existing algorithms. There are two ways in which a geometric hierarchy should lead to improvements in existing algorithms. The first is by reducing the number of comparisons needed to sort objects and the second is by eliminating from potential consideration an entire portion of the environment because an object obscures it.

Since sorting is the central problem of visible surface algorithms, the performance of these algorithms improves with improved sorting methods. Indeed, many of the fast visible surface algorithms that have been discussed have resulted from clever utilization of image-space coherences, such as scan-line coherence, to improve sorting speeds. In the present hierarchical framework, the geometric proximity of the subobjects of an object provides an object-space coherence that can also be utilized to decrease sorting time.

For example, consider an ideal case of a binary tree as shown in Figure 3. Each node of the tree has associated with it a bounding volume, but since this ideal tree is the result of clipping, only the terminal nodes actually represent geometric primitives, e.g. polygons or patches. Assuming that there are n levels in the tree, not counting the root node, there are $m = 2^n$ terminal nodes.

If the structure is ignored, then the fastest possible sort of these terminal nodes is accomplished with proportional to $m \log_2 m$ comparisons using a quicksort. However, if the structure is utilized and if the bounding volumes of siblings do not overlap, which is admittedly an optimum arrangement, then the number of required operations is $p2^0$ for the first level, $p2^1$ for the second level, $p2^2$ for the third, etc., where p is a proportionality factor. Summing the number of operations performed at all levels yields $p\sum_{i=0}^{n-1} 2^i = p(2^n - 1)$, or roughly pm. In other words, by using the structure, in the optimum situation of no overlap, the sorting time grows linearly rather than as $m \log_2 m$.

Fig. 3. An ideal binary hierarchy in which none of the terminal nodes overlap. The first $p2^0$ comparison sorts all objects into two classes, the second $p2^1$ comparisons sort them into 4 classes, etc. Summing all comparisons from all levels yields $p(2^n - 1)$ comparisons.



Of course, this analysis holds only for a binary hierarchy in which none of the siblings' bounding volumes overlap, which is an idealized situation. A binary hierarchy might not be appropriate, and any complex environment will no doubt have some overlap, although presumably not a very large amount. However, the point here is that sorting methods which utilize the geometric structure can yield a considerable performance improvement over those which do not, even under less than ideal conditions.

The other improvement provided by a deeply structured geometric hierarchy is that of eliminating a potentially large part of the structure from consideration because an object obscures it. Such an improvement requires defining for each object (in the generic sense) both a simple occluded volume, Δ, such that if Δ is obscured then the entire object is obscured, and a simple occluding volume, δ, such that if δ obscures something then that thing is sure to be obscured by the object. Clearly, Δ exists for all objects, whereas δ might not exist for some objects, such as an open-ended cylinder or a transparent object. Δ can be just the bounding sphere used in clipping, but δ is in general additional information that must be kept for each object.

Recursive descent, visible surface algorithm. The above considerations suggest a totally new recursive-descent visible surface algorithm in which at each level all objects are sorted according to their bounding volumes. If any of the bounding volumes overlap both laterally and vertically then the occlusion test potentially allows one (or more) of the objects, and hence all of its descendents, to be totally eliminated from consideration.

Using the ordering thus obtained, the same sorting and occlusion tests are recursively applied to each of the descendants of these objects; in those cases where two or more objects' bounding volumes overlap in all three dimensions, indicating potential intersections, the descendents of these objects are treated as if they have the same parent nodes at the next level of recursion. Of course, recursion terminates when a terminal node is reached, and the net result of descending the tree is a very rapid sort of the primitives represented by these terminal nodes. Under ideal conditions, the computation time of this algorithm grows linearly with the *visible* complexity of the scene.

Since both this algorithm and the clipping algorithm described above recursively descend a tree structure, it seems natural to combine them. Doing so not only potentially eliminates area tests on occluded objects but also potentially decreases the size of the working set. If all processing is performed by a single pro-

cessor, such as a general purpose computer, then the algorithms are probably most conveniently integrated into a single algorithm. However, if multiple processors are available, whether special purpose hardware or general purpose computers, then the algorithms might be left separate or combined according to whether parallelism is achieved by pipelining or otherwise.

Building structured databases. Obtaining a good graphical database is a very time consuming and difficult part of computer picture research. Databases obtained by careful measurement of real objects, by "building" objects from collections of simple mathematical objects, or by sculpturing surfaces in three dimensions [4] are at least as valuable as the visible surface algorithms that render them.

At first glance it appears that the structural framework multiplies the dimensions of this problem since multiple descriptions of the same object must be defined. However, in the case of carefully measured real objects, the multiple descriptions can be produced by judicious "bottom-up" pruning of existing definitions of the objects in their most detailed form. Therefore use can be made of all objects that have already been defined.

Those existing objects modeled with surface patches also present no problem. The coarser, high-level descriptions of these objects can be obtained by replacing the patches themselves with polygons and proceeding with the "bottom-up" pruning mentioned above to obtain even coarser descriptions. The finer, low-level descriptions of the objects can be obtained by "top-down" splitting of the surface patches, as in the Catmull algorithm. This can be done either at display time or beforehand in building the database; the difference is the traditional time/space tradeoff.

4. Conclusions

All of the recent major advances in computer picture research have resulted from either explicitly or implicitly incorporating structure information in the geometric modeling techniques. This research represents an attempt to encompass all of these advances in a more general structural framework as a unified approach to solving a number of the important problems of systems for producing computer pictures.

The proposed hierarchical models potentially solve a number of these problems. They provide a meaningful way to vary the amount of detail in a scene both according to the screen area occupied by the objects in the scene and according to the speed with which an object or the camera is moving. They also extend the total range of definition of the object space and suggest convenient ways to rapidly access objects by utilizing a graphical working set to accomplish frame coherence.

An important aspect of the hierarchical models is that by providing a way to vary detail they can yield an incremental improvement to existing systems for producing computer pictures without modifying their visible surface algorithms. Another incremental improvement is then possible by incorporating the structure in the sorting phases of existing algorithms. A final improvement is suggested by a totally new recursive descent visible surface algorithm in which the computation time potentially grows linearly with the visible complexity of a scene rather than as a worse than linear function of the object-space complexity.

References

1. Bouknight, W.J. A procedure for generation of three-dimensional half-toned computer graphics representations. *Comm. ACM, 13*, 9 (Sept. 1970), 527.
2. Computer Aided Operations and Research Facility, U.S. Maritime Service Simulator (principal contractor Philco-Ford, visible-surface processor by Evans and Sutherland Comptr. Corp.).
3. Catmull, E. A subdivision algorithm for computer display of curved surfaces. Tech. Rep. UTEC-CSc-74-133, U. of Utah, Salt Lake City, Utah, Dec. 1974.
4. Clark, J.H. 3-D design of free-form B-spline surfaces. UTEC-CSc-74-120, Ph.D. Th., U. of Utah, Salt Lake City, Utah, (abridged version Designing surfaces in 3-D. *Comm. ACM 19*, 8 (Aug. 1976), 464–470.)
5. Coons, S.A. Surfaces for computer-aided design of space forms. Project MAC TR-41., M.I.T., Cambridge, Mass., June 1967.
6. Crow, F.C., and Bui-Tuong Phong. Improved Rendition of Polygonal Models of Curved Surfaces. Proc. Second USA-Japan Comptr. Conf., Aug. 1975, p. 475.
7. Csuri, C. Computer animation, Computer Graphics *9*, 1 (1975), 92–101 (Issue of Proc. Second Ann. Conf. Comptr. Graphics and Interactive Techniques).
8. Denning, P.J. The working set model for program behavior. *Comm. ACM, 11*, 5 (May 1968), 323–333.
9. Electonic scene generator expansion system. Final Rep., NASA Contract NAS 9-11065, Defense Electronic Div., General Electric Corp., Syracuse, N.Y., Dec. 1971.
10. Gouraud, H. Computer display of curved surfaces. *IEEE Trans. Computers C-20* (June 1971), 623.
11. Nasa-Ames Short Take-off and Landing Simulator (built by Evans and Sutherland Comptr. Corp.).
12. New York Inst. Tech., Comptr. Animation Dep.
13. Newell, M. The utilization of procedure models in digital image synthesis. Ph.D. Th., Comptr. Sci., U. of Utah, Salt Lake City, Utah, 1975.
14. Newell, M.E., Newell, R.G., and Sancha, T.L. A new solution to the hidden-surface problem. Proc. ACM 1972 Ann. Conf., pp. 443–448.
15. Bui-Tuong Phong. Illumination for computer generated pictures. *Comm. ACM 18*, 6 (June 1975), 311–317.
16. Rediflow Flight Simulation, Ltd., NOVOVIEW Visual Systems (video system provided by E&S Comptr. Corp.).
17. Riesenfeld, R.E. Applications of B-spline approximation to geometric problems of computer aided design. Ph.D. Th., Syracuse U., Syracuse, N.Y., 1972.
18. Schumacker, R.A., Brand, B., Gilliland, M., and Sharp, W. Study for applying computer-generated images to visual simulations. AFHRL-TR-69-74, US Air Force Human Resources Lab., Washington, D.C., Sept. 1969.
19. Sutherland, I.E. Sketchpad: a man-machine graphical communication system. TR 296, M.I.T Lincoln Labs, M.I.T., Cambridge, Mass., Jan. 1963.
20. Sutherland, I.E., Sproull, R.F., and Schumacker, R.A. A characterization of ten hidden-surface algorithms. Computing Surveys, *6*, 1 (March 1974), 1–55.
21. Watkins, G.S. A real-time visible-surface algorithm. UTECH-CSc-70-101, Ph.D. Th., Comptr. Sci. Dep., U. of Utah, Salt Lake City, Utah, June, 1970.
22. Wipke, T., et al. *Computer Representation and Manipulation of Chemical Information.* Wylie Interscience, New York, 1974.
23. Wylie, C., Romney, R.S., Evans, D.C., and Erdahl, A. Halftone perspective drawings by computer. Proc. AFIPS 1967 FJCC, Vol. 31, AFIPS Press, Montvale, N.J., pp. 49–58.

ON VISIBLE SURFACE GENERATION BY A PRIORI TREE STRUCTURES*

Henry Fuchs
University of North Carolina at Chapel Hill
Zvi M. Kedem
The University of Texas at Dallas
Bruce F. Naylor
The University of Texas at Dallas

ABSTRACT

This paper describes a new algorithm for solving the hidden surface (or line) problem, to more rapidly generate realistic images of 3-D scenes composed of polygons, and presents the development of theoretical foundations in the area as well as additional related algorithms. As in many applications the environment to be displayed consists of polygons many of whose relative geometric relations are static, we attempt to capitalize on this by preprocessing the environment's database so as to decrease the run-time computations required to generate a scene. This preprocessing is based on generating a "binary space partitioning" tree whose inorder traversal of visibility priority at run-time will produce a linear order, dependent upon the viewing position, on (parts of) the polygons, which can then be used to easily solve the hidden surface problem. In the application where the entire environment is static with only the viewing-position changing, as is common in simulation, the results presented will be sufficient to solve completely the hidden surface problem.

BSP-tree

INTRODUCTION

One of the long-term goals of computer graphics has been, and continues to be, the rapid, possibly real-time generation of realistic images of simulated 3-D environments. "Real-time," in current practice, has come to mean creating an image in 1/30 of a second--fast enough to continually generate images on a video monitor. With this fast image generation, there is no discernable delay between specifying parameters for an image (using knobs, switches, or cockpit controls) and the image's appearance on the monitor's screen. Systems which can achieve this kind of performance are currently so expensive ($1M and up) that very few users can afford them. Users with more modest budgets have to be content with severely more limited performance--either a lower quality image ("wire frame" instead of solid-object modeling) or slower interaction (a time lag of several seconds to several minutes for a solid-object image).

PROBLEM STATEMENT

The problem to be solved is:

Given

1. a data base describing a 3-D environment in terms of, say, a few thousands tiles (polygons) describing the surfaces of the various objects in the environment, one or more light sources and
2. the (simulated) viewing position, orientation, and field of view,

Generate a color video image of the environment as it would appear from the given viewing position and orientation.

This image generation task consists, broadly, of the following three steps:

1. transforming points into the image space,
2. clipping away polygons outside the field of view,
3. generating the image from the polygons that remain. Generating the image consists of determining the proper color (intensities of red, green, and blue) for each of perhaps 250,000 picture elements (approximately 500 rows of dots, with 500 dots in each row). For each picture element ("pixel"),
   a) find the polygon closest to the viewing position. (This will be the visible polygon at this pixel, the

*This research was partially supported by NSF under Grants MCS79-00168 and MCS79-02593, and was facilitated by the use of Theory Net (NSF Grant MCS78-01689).

polygon which obstructs all others.)

b) given the visible polygon, determine the proper color for the pixel by evaluating a lighting model formula, see, e.g., (Newman and Sproull, 1979).

PROPOSED SOLUTION

Since current moderately-priced ($40-80k) real-time line-drawing systems (e.g. Evans and Sutherland Picture System 2, Vector General Model 3404) can easily perform steps 1 and 2, we shall concentrate on solutions to step 3. New solutions to this remaining step could then be combined with already available solutions to produce a complete system. Further, we believe step 3b can be effectively solved by distributing the individual pixel calculations among many small processors (Fuchs and Johnson, 1979). We thus concentrate in this paper on step 3a, determining the visible polygon at each pixel.

We propose an alternative solution to an approach first utilized a decade ago (Schumaker et al., 1969) but due to a few difficulties, not widely exploited. The general approach is based on the observation that in a wide variety of applications many images are generated of the same environment with only a change in the viewing position and orientation, but no change in the environment. For example, pilots in a simulator may practice many different landings at the same airport, with each landing generating thousands of new images. Similarly, an architect may "walk" through a newly designed house or housing development; a biochemist may rotate or move about a complicated protein molecule. To take advantage of such static environments, the data base is preprocessed once (for all time, or until the data base is changed) before any images are generated. In this preprocessing stage, certain geometric relationships are extracted which can then be used to speed up the visible polygon determination for each pixel, for all possible images.

It is important to note that although the development here is given only rigid objects and environments, these concepts can be extended to handle environments with some moving objects.

SOLUTION OVERVIEW

In order to determine the visible surface at each pixel, traditionally the distance from the viewing position to each polygon which maps onto that pixel is calculated. Most methods attempt to minimize the number of polygons to be so considered. Our approach eliminates these distance calculations entirely. Rather, it transforms the polygonal data base (splitting polygons when necessary) into a binary tree which can be traversed at image generation time to yield a visibility priority value for each polygon. These visibility priorities are assigned in such a way that at each pixel the closest polygon to the viewing position will be the one with the highest visibility priority. As we shall see, the visibility priorities are a function of the viewing position; they remain constant for all pixels in every image generated from the same viewing position. In cases for which these visibility priority numbers cannot be assigned to the original polygons (see, e.g., fig. 6) and some polygons need to be split, the splitting is done only once -- during the preprocessing phase -- never at image generation time.

PREPROCESSING PHASE

Let us now consider the set of polygons $P = (p_1, p_2, \ldots, p_n)$ which define the 3-D environment. Choose an arbitrary (for now) polygon p_k from this set. We note that the plane in which this polygon lies partitions the rest of 3-space into two half-spaces--call these S_k and $S_{\bar{k}}$. The two half-spaces are identified with the positive and negative sides of the polygon p_k. If p_k was defined with a "front" side, then that side is considered as the positive one; otherwise, one of the sides is arbitrarily chosen at this time to be the positive side.

What can we say about visibility priorities of these polygons? We know that if the viewing position is in one half-space, say in S_k, that no polygon within $S_{\bar{k}}$ can obstruct either polygon p_k or any polygon in S_k (see figure. 1).

Therefore, we split each of the polygons in $P - \{p_k\}$ along the plane of p_k, putting the polygons (or parts of them) which lie in S_k into one set and polygons which lie in $S_{\bar{k}}$ into another set. (Polygons coplanar with p_k can be put into either set.) We can represent the results of this splitting process by a binary tree (we'll call it a Binary Space Partitioning, or "BSP" tree) in which the root contains p_k and each branch's subtree contains the set of polygons associated with one of the half-spaces (Fig. 2).

We next consider one of the two new sets of polygons, say the one in S_k. We remove a polygon, say p_1 and split the remaining polygons in S along the plane of p_1, putting those polygons (or parts thereof) lying on the positive side in one

set ($S_{k,j}$) and those lying on the negative side in another set ($S_{k,-}$). The overall tree after this step is shown in Fig. 3.

To complete the construction of the BSP tree we continue splitting sets until no non-null sets remain.

The entire preprocessing phase, then, consists of transforming the entire polygonal data base into a BSP tree by the following recursive procedure (stated in a simple pseudo-PASCAL):

```
PROC Make_tree(pl:polygon_list): tree;

BEGIN
k=Select_polygon (pl);
pos_list := null; neg_list := null;
/* pos refers to positive parts
   neg refers to negative parts */
FOR i := 1 TO Size_of(pl) DO
  BEGIN
  IF i <> k THEN
   BEGIN
   Split_polygon(pl[i], pl[k],
                 pos_parts,neg_parts);
   Add (pos_parts, pos_list);
   Add (neg_parts, neg_list)
   END
  END;
RETURN    Combine_tree(Make_tree(pos_list),
                       pl(k),
                       Make_tree(neg_list))
END;
```

We note again that this process is only performed once for all possible images from all viewing positions; the tree remains valid as long as the scene doesn't change.

IMAGE GENERATION PHASE

Calculating the visibility priorities, once the viewing position is known, is a variant of an in-order traversal of the environment's BSP tree (traverse one subtree, visit the root, traverse the other subtree). We wish, for example, to have an order of traversal that visits the polygons from those farthest away to those closest to the current viewing position. At any given node, there are two possibilities: positive side subtree, node, negative side subtree or negative side subtree, node, positive side subtree. We choose one of these two orderings based on the relationship of the current viewing position to the node's polygon. Specifically, we are interested in the side (positive or negative) of the node's polygon where the current viewing position is located. Let's call the two sides the "containing" side and the "other" side. The traversal for a back-to-front ordering is 1) the "other" side, 2) the node, and 3) the "containing" side. (This side-of-node-polygon determination is, of course, just a check of the sign of the Z component of the node polygon's normal vector after the usual transformation to the screen coordinate system.)

This notion of a traversal may be embodied in at least two different ways for visible surface image generation. One alternative is to assign priorities to polygons in the order that we visit them. Using the traversal order just described we will get a low-to-high visibility priority assignment. These values can then be used within a conventional visible surface display algorithm wherever visibility determinations need to be made. The other obvious alternative, which in fact is the one that we have implemented, does not assign explicit visibility priority values to polygons but uses the traversal to drive a "painter's" algorithm which paints onto the screen's image buffer each polygon as it is encountered in the traversal. Since higher priority polygons are visited later in the traversal and thus painted later, they will overwrite any overlapping polygons of lower priority. The following recursive procedure generates a visible surface image in the above-described manner.

```
PROC Back_to_front(eye:viewing_position;
                   t: BSP_tree);

BEGIN
IF Not_null (t) THEN
  IF pos_side_of (root [t],eye)
  THEN

    BEGIN
    Back_to_front (eye, neg_branch [t]);
    Display_polygon (root [t]);
    Back_to_front (eye, pos_branch[t])
    END

  ELSE

    BEGIN
    Back_to_front (eye, pos_branch[t]);
    Display_polygon (root[t]);
    Back_to_front (eye, neg_branch[t])
    END
END
```

Figures 4,5,and 6 illustrate this visible surface algorithm. Since the display used had only one bit per pixel, the procedure Display_polygon painted the interior of the polygon the background shade and painted the outline of the polygon in the other shade.

The possible weakness of this approach is that the number of polygons in the tree may increase sharply. (Recall, every root polygon splits all crossing polygons in its list in order to put any polygon in one or the other of its

subtrees.) We have attempted to limit this increase by selecting the root polygon at each stage to be the one whose plane splits the minimum number of polygons in its list. Table 1 indicates the performance of the system in limiting the number of polygons in the BSP tree. Figures 7 and 8 show the BSP tree for the environments of Figures 4 and 6, respectively.

Fig. no.	No. of Original Polygons	No. of Polygons in BSP Tree
4	11	11
5	72	100
6	3	5

Table 1: Number of polygons in tree versus original data base

We are currently examining a more sophisticated strategy for minimizing the number of polygons in the BSP tree. In addition to the just-described criterion of choosing a node polygon as one that minimizes the number of polygons that are split, a second criterion is also considered. This one maximizes the number of "polygon conflicts" eliminated. We define a polygon conflict as an occurrence between two polygons in one list in which the plane of one polygon intersects the other polygon. The hope is that these eliminated polygon conflicts will reduce the number of polygons which will need to be cut in the descendant subtrees. More precisely, if P is the set of polygons, then form the sets S_1, S_2, S_3 for each polygon p P as follows:

$S_1 \equiv \{q \in P \mid q$ is entirely in the positive half space of p$\}$

$S_2 \equiv \{q \in P \mid q$ is intersected by the plane of p$\}$

$S_3 \equiv \{q \in P \mid q$ is entirely in the negative half-space of p$\}$

We define a function

$$f(s_i, s_j) = \begin{cases} 1; & \text{polygon } s_j \text{ and the plane of } s_i \text{ intersect} \\ 0; & \text{otherwise} \end{cases}$$

and

$$I_{m,n} = \sum_{s_i \in S_m} \sum_{s_j \in S_n} f(s_i, s_j)$$

We then select the p such that for $S_1(p), S_2(p), S_3(p)$ the expression $[I_{1,3} + I_{3,1} - (|S_2| * \text{weight})]$ which is maximal.

FORMAL DEVELOPMENT

Let us now examine the nature of th binary space-partitioning (BSP) tree mor closely. The construction can be carried in essentially identical manner, for an dimension; nonetheless, it is only th three-dimensional version that is of majo interest to us here. However, it i easier to explain its nature in the two dimensional setting, as the variou geometric structures arising can b clearly drawn; thus the discussion of th properties of the tree will be presente assuming a two-dimensional universe. Nonetheless, we encourage the reader, a the next section of the paper is read, t extrapolate the three-dimensiona interpretation. In the latter portion o the paper, where combinatorial issues ar examined, the results will be given fo both two and three dimensions, since combinatorial complexity is dimensio dependent. We now begin with some (slightly non-standard) terminology.

Segment - an oriented closed convex subset of a line, i.e., a finite segment, a ray, or a line, with a direction associated with it.

Region - a closed convex set of points of a plane. (A region is normally defined as an open connected set.)*

Extension of a Segment in a Region - given a segment s and a region R, define the extension of s in R to be the intersection of the line on which s lies with the region R, obtaining the segment s_R. Assign to s_R the direction induced by s (we indicate this by pointing an arrow to the right).

Note that a region can be unbounded (a plane) "partially bounded" (e.g., a half-plane), or (completely) bounded (e.g., a finite polygon). The motivation for defining regions and segments in this manner is that in general we have no interest in distinguishing between the bounded, partially bounded, and unbounded sets. The 3-space analogies to segments,

*A set is open if there is an "implicit boundary" which is not in the set. Formally, a set of points R in the plane is open if and only if $\forall x \in R$, $\epsilon > 0$ such that $\forall y[|x-y|<\epsilon \Rightarrow y \in R]$. A closed set is the complement of an open set (if bounded, the set includes the boundary). Formally, a set of points R in a plane is closed iff for every converging sequence $x \to x$, $\forall n[x \in R \Rightarrow x \in R]$.

and regions are polygons (or alternately, regions) and sectors (or volumes) respectively. The orientation of the polygons corresponds to the usual notion of the front and back sides. We are now ready to examine the general algorithm for construction of a labeled binary space-partitioning tree.

Algorithm I: Construction of a (2-space) BSP tree

Input - a region R and a set of segments Σ lying in R

Output - A BSP Tree

Method - call the function, BSPT, with R and Σ as parameters and $\hat{\Sigma} \leftarrow \phi$.

Procedure - BSPT (R:region; Σ:set of segments) :node

Begin

If $\Sigma \neq \phi$ then

begin

choose $s \in \Sigma$ and form $\hat{s}_R$

$\hat{\Sigma} \leftarrow \hat{\Sigma} \cup \{ \hat{s}_R \}$

Partition R and Σ by $\hat{s}_R$ into R_s, $R_{\bar{s}}$, Σ_{R_s}, $\Sigma_{R_{\bar{s}}}$ defined as:

$R_s \equiv \{ p \in R \mid p \in \hat{s}_R$ or p lies to the right of $\hat{s}_R \}$

$R_{\bar{s}} \equiv \{ p \in R \mid p \in \hat{s}_R$ or p lies to the left of $\hat{s}_R \}$

$\Sigma_{R_s} \equiv \{ B \cap R_s \mid B \in \Sigma - \{s\} \}$

$\Sigma_{R_{\bar{s}}} \equiv \{ B \cap R_{\bar{s}} \mid B \in \Sigma - \{s\} \}$

Create a new node v

leftson(v) :=BSPT($R_{\bar{s}}, \Sigma_{R_{\bar{s}}}$)

rightson(v) := BSPT (R_s, Σ_{R_s})

label(v) := s_R

return(v)

End

Else

Create a leaf l

label(l):=R

return(l)

End BSPT

Let us look at an example before examining the properties of this algorithm.

Let R be a square and $\Sigma = \{ a,b,c \}$, as in figure 9a.

If a is chosen first, we get figure 9b which creates figure 9c. If, next, $b_{R_{\bar{a}}}$ is chosen before c, the final result will appear as in figure 10.

Consider now the set $\hat{\Sigma}$ of segments, which of course lies wholly within the original R. It is easily seen that it partitions R into convex regions (polygons). Each such region, together with its boundary, will be referred to as an area (volume for 3-space). The set of all the areas created by the algorithm will be referred to as a tessellation. The areas may be thought of as the intersection of half-planes (half-spaces for 3-D) created by the lines on which the elements of Σ (or $\hat{\Sigma}$) lie. The purpose of orientation of the segments is to distinguish between the two half-planes. The subscripts of each region, generated by algorithm I, indicate the half-planes whose intersection forms the region. As an example, refer to figure 11 which is a BSP tree for the tessellation in fig. 10 where parentheses are used to indicate subscripting of regions.

CHARACTERISTICS OF A BSP TREE

It should be clear by now that the algorithm performs a recursive partitioning of the plane by the segments lying in it. However, observe that given a set of segments, that more than one tessellations can be generated by the algorithm depending upon the order in which segments are selected. Observe that in fig. 9, had the order of selection been c, b, a, fig. 12 would have been produced, which not only looks different from fig. 10, but has four areas, as opposed to five. Since a tessellation is formed by the extended segments, as opposed to the segments themselves and the length of an extended segment is dependent on the size of the region containing it at the time it is extended, selecting segments in different orders produces different regions, and thus the dependence of the tessellation on the order of selection.

It is also possible to have, for a given set of segments, more than one tree which describes the same tessellation. Assume that at some stage of the construction of the tree, we are examining the region R_k and the associated set of segments $\Sigma_{R_k} = \{s_1, s_2, \ldots, s_m\}$. If $\bigcup_{i=1}^{m} s_i = \bigcup_{i=1}^{m} \hat{s}_i$ with respect to R_k, then every permutation π on i= 1,2,...,m will result in a different subtree, where the

subtree is generated by selecting segments in the order $s_{\pi(1)}, s_{\pi(2)}, \ldots, s_{\pi(m)}$. Nonetheless, every subtree will describe the same tessellation of R_k. Consequently, there are distinct trees describing the same tessellation of the original region R. For example, in figure 13, either tree specifies the same tessellation.

An important special case occurs when the initial set of segments is equivalent to the extended set, i.e., $\cup\{s|s\in\Sigma\} = \cup\{\hat{s}|\hat{s}\in\hat{\Sigma}\}$. If in addition the initial region is a plane, all of the elements of Σ, would be lines. Since extension has no effect, the tessellation is fixed before the algorithm begins. We call such a tessellation a maximum tessellation because any set of segments lying on the same set of lines can produce only tessellations whose areas are the union of the areas of the maximum tessellation, as can be seen by comparing figures 10 and 12 with fig. 14. It follows that any set of segments has a corresponding maximum tessellation whose cardinality is the maximum of the number of areas produced by any tessellation resulting from the set. In general, the number of different tessellations that can be derived from a set Σ is, in some sense, the complement of the number of distinct trees which describe the same tessellations.

A BSP tree constructed by algorithm I contains nodes labeled with segments and nodes labeled with areas. The segment nodes are exactly the interior nodes of the tree and the "area" nodes are the leaves. The algorithm can be thought of as first generating a binary tree composed of only the segment nodes. There will then be segment nodes which have one or two empty sons. (Every node of a binary tree has potentially two sons, left and right. If a node does not have one or both sons, we refer to these as "empty sons.") At each empty son, an area node is added. The resulting tree is such that all segment nodes have both a left and a right son, either of which could be another segment node or an area node. Since binary trees of n nodes have n+1 empty sons, it follows that the number of area nodes is one more than the number of segment nodes, thus a tree of 2n-1 nodes is needed to represent a tessellation containing n areas.

Each subtree of a BSP tree represents some region R_i in the sense that the union of all the areas represented by the leaves or R_i equals R_i (the segments represented by the segment nodes of R_i are thus, also included in this union). For notational purposes we will designate the region represented by the entire tree as R_0. This, of course, is the original region from which the tessellation is formed. The extension of the segment s represented by the root q of a subtree partitions a region R_i, and the regions represented by the two subtrees of q are the two half-spaces formed from R_i by $\hat{s}$. If, upon traversing the tree one reaches q, then taking the left or right branch of q would have a geometric correspondence to selecting one of these two half-spaces. A path in the tree, then, reflects a successive selection of smaller and smaller portions of R_0. In fact the region represented by a subtree is the intersection of the half-spaces with respect to R_0 formed by the extension of the segments which are on the path to the root of the subtree q (but not including q). It immediately follows that the area which is "added" at each empty son is exactly the intersection of the half-spaces with respect to R_0 formed by the extension of segments whose nodes are on the path to the son.

It is easy to see how a BSP tree can be used to locate which area of the tessellation a point lies. Beginning at the root, determine on which side of a segment the point lies and proceed to the son representing the half-space corresponding to that side (points on the line being assigned arbitrarily to one of the two half-spaces). Repetition of this process will generate a path to a leaf node that represents the area in which the point lies, thus solving what might be called the "location problem" with respect to a tessellation.

BSP Tree Used for Priority Ordering

The ability of a BSP tree to be used for the generation of a priority ordering is based upon the principle that given in which half-space lies the point to which the ordering is relative (usually thought of as the "eye" or viewing position), all points in this same half-space will have priority over all points in the other half-space. Although this fact is fairly self evident for half-spaces, it is also true for any two convex regions.

To obtain a priority ordering from the tree, an inorder traversal is performed. The choice of taking the left or right branch of a node q representing segment s is always made in favor of the subtree which represents the region that is contained in the same half-space that the viewing position is in, this half-space having been formed by $\hat{s}$ with respect to R_0. It is easy to see that such a policy will result in the first area node to be reached being the one in which the viewing position lies, i.e. the solution to the location problem mentioned earlier. Priority is assigned to a node upon backing-up from it during the traversal.

Thus for each node q, all nodes of the chosen subtree receive higher priorities than q, and similarly, all nodes of the remaining subtree obtain a lower priority than q. The entire traversal of the tree will then produce a total ordering of the nodes, and this is precisely the visibility priority of the elements represented by the nodes. Note that it is requisite that R_0 be convex to guarantee this property. Since the partitioning of a convex object produces two convex objects, the convexity of R_0 implies the same property for all subsequent refinements of R_0 during the construction of the tree. Thus all areas are convex which is sufficient to guarantee the existence of a priority ordering of the areas.

Comparison of Uses of the BSP Tree

The first appearance of a BSP tree in the general literature was in Sutherland, et al. (1974) reviewing the work of Schumaker, et al. (1969), although the tree was not named and its general properties were not developed. The application presented was that in which invisible "dividing planes" were introduced to the data base. The method involved the designer of a simulation scene manually positioning "clusters" such as buildings, trees, mountains, etc., so that vertical dividing planes could be placed between the clusters to varying extents. This resulted, in terms of a BSP tree, in the generation of a tessellation of the surface by the dividing places which are represented by segment nodes, and each cluster was contained wholly within an area. Thus each cluster corresponded to an area node. A priority ordering could then be obtained on the clusters.

Additional power is available if the tessellation is a maximum tessellation. In this case, it is possible to compute off-line the priority ordering for each case of the viewing position being in a different area. This follows from the fact that since the areas are formed by a maximum tessellation, it is not possible for two different points in the same area to be on different sides of the extension of a segment with respect to R_0 (since in a maximum tessellation all segments are equal to their extensions with respect to R_0). Thus for each area the traversal of the tree is fixed. The Sutherland, et al., presentation suggests taking advantage of this by pre-computing and storing for each area its inherent priority ordering on the clusters (since the dividing planes are not part of the scene they need not be included in the ordering). It was then sufficient to solve the location problem in order to obtain the priority ordering. Since this method requires n^2 storage space (where n is the number of clusters) and the traversal of the tree is O(n), it is not clear whether this approach is advantageous. Also since a maximum tessellation is required the tree will be the largest possible for a given set of clusters.

The application of BSP trees introduced in this paper is something of a complement to that presented in Sutherland, et al. Here those objects represented by the segment (or polygon) nodes constitute the visible data while the areas of the tessellation are of no importance. In fact, the function Make_tree presented earlier produces only the segment nodes. The area nodes are only implied by the empty sons. Also, Make_tree forms a BSP tree for three dimensions while the former method, although working in 3-D, forms a BSP tree for two dimensions, and Make_tree's tessellation in general is not maximal. Clearly the BSP tree can be used with dividing planes to divide 3-space into volumes, and a hybrid of polygons and dividing planes could also be developed. For instance, each area node of a tree constructed with dividing planes could be replaced with a BSP tree constructed of polygons for the cluster contained in the area.

Combinatorics of the BSP Tree

We will now examine the size of the BSP trees. The previous discussion was presented, for simplicity's sake, for the 2-D case; here we will derive some formulas both for the 2-D and 3-D BSP trees. Although we are most interested in the 3-D case, 2-D is important in the special 3-D case in which all of the objects "sit" on the ground and can be separated by vertical planes. This is equivalent to a 2-D BSP tree corresponding to the 2-D scene obtained by projecting the objects and the separating plane on the ground plane.

As noted previously, the BSP tree can be created by both infinite and finite objects. The infinite objects are planes for the 3-D case are lines for the 2-D case. The corresponding finite objects are non-intersecting convex polygons and segments. We will examine these two extremal cases in turn.

A d-dimensional BSP tree partitions the d-dimensional space by (d-1)-dimensional objects. We thus examine first, what is the maximum number $f_d(n)$ of volumes of a d-dimensional space that can be created by n (d-1)-dimensional planes. In the 2-D case we have been considering, this corresponds to the maximum tessellation of the plane using lines.

The general formula is

$$f_d(n) = \sum_{i=0}^{d} \binom{n}{i}.$$

As there is a one-to-one correspondence between the volumes and the leaves of the binary BSP tree, the number of the nodes of the BSP tree is $2f_d(n)-1$.

How many (d-1)-dimensional regions created from (d-1)-dimensional planes under the assumption that no 3 planes intersect along a single (d-2)-dimensional line? It can be shown that the number is

$$\sum_{i=0}^{d} i \binom{n}{i}.$$

In the other extremal case, where the objects given are n (d-1)-dimensional non-interpenetrating convex polygons, we examine the worst case, namely compute the maximum number of the (d-1)-dimensional regions that are obtained from the polygons by the intersection of the n (d-1)-dimensional planes on which they lie. It can be shown that the number is

$$\binom{n}{2} + n^{d-1}$$

We summarize the results in Table 2 for the two interesting cases d=2 and d=3.

	Volumes	Unbounded Objects	Bounded Objects
2-D:	$\frac{n^2+n}{2}+1$	n^2	$\frac{n^2+n}{2}$
3-D:	$\frac{n^3+5n}{6}+1$	$\frac{n^3-n^2+2n}{2}$	$\frac{n^3+3n^2+2n}{6}$

Table 2: Maximum possible nodes in BSP tree.

Conclusion

A solution has been presented to the visible surface problem which appears to be more efficient than previous solutions whenever many images are to be generated of the same static environment. The algorithm is easy to implement since both phases, the preprocessing and the image generation, can each be succinctly stated in a short recursive procedure. The major potential weakness, a large increase from the number of original polygons in the data base to the number in the BSP tree, has not occurred in any environment so far encountered.

Acknowledgement

We wish to thank Greg Abram for much needed and appreciated program development, Mike Cronin for numerous improvements to the narrative, and the referees for helpful and thorough reviews of the first draft of this paper.

References

Berman, G. and Fryer, K.D. Introduction to Combinatorics, (1972) Academic Press.

Fuchs, H. and Johnson, B. "An Expandable Multiprocessor Architecture for Video Graphics" Proc. 6th Annual Symp. on Computer Architecture, (1979) pp 58-67

Schumaker, R.A., Brand, P., Gilliland, M. and Sharp, W. "Study for Applying Computer-Generated Images to Visual Simulation," AFHRL-TR-69-14, U.S. Air Force Human Resources Laboratory (1969)

Sutherland, I.E., Sproull, R.F. and Schumaker, R.A. "A Characterization of Ten Hidden-Surface Algorithms", (1974) ACM Computing Surveys, 6 (1): 1-55

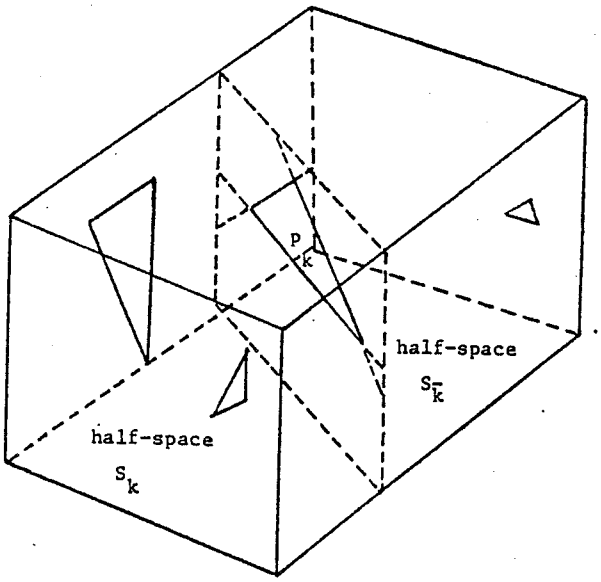


Figure 1: Environment split by plane of p_k

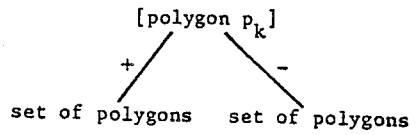


Figure 2: Beginning of BSP tree construction

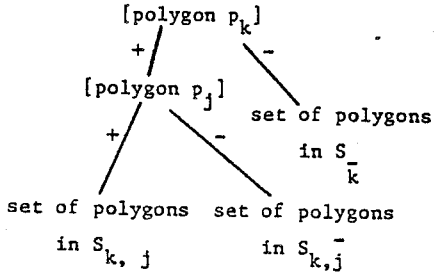


Figure 3: BSP tree after two steps

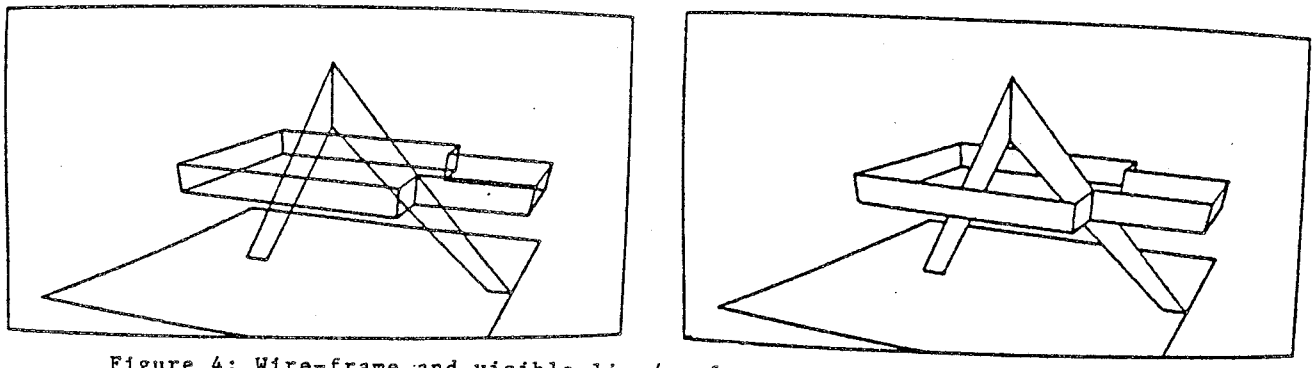


Figure 4: Wire-frame and visible line/surface images of same environment
(11 original polygons; 11 in BSP tree)

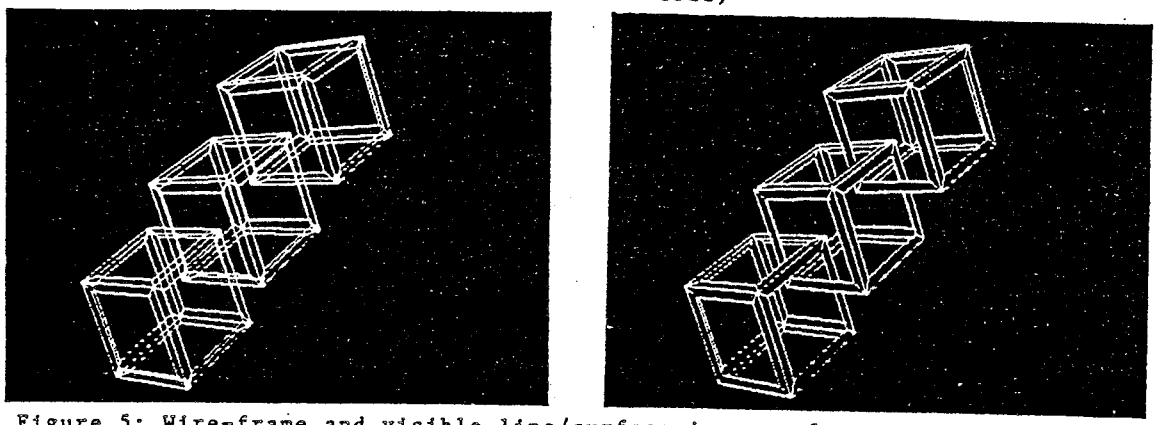


Figure 5: Wire-frame and visible line/surface images of same environment
(72 original polygons; 100 polygons in BSP tree)

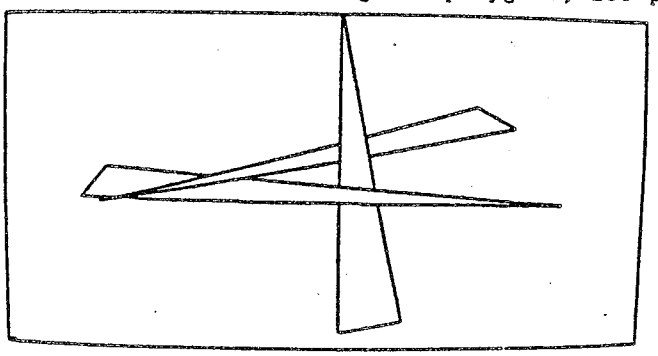
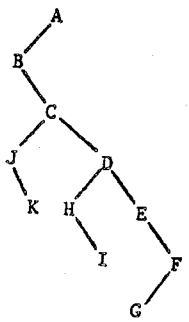
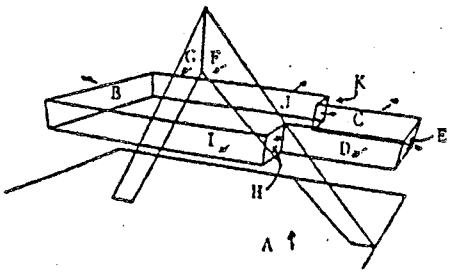


Figure 6: Visible line/surface image of simple object whose polygons cannot be directly assigned visibility priorities (some polygons here have been split during preprocessing)

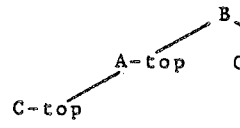


(left branches are positive)



(arrows on positive side of polygons)

Figure 7: BSP tree and polygons of Fig.4



(left branches are positive;
positive sides of all polygons are visible)

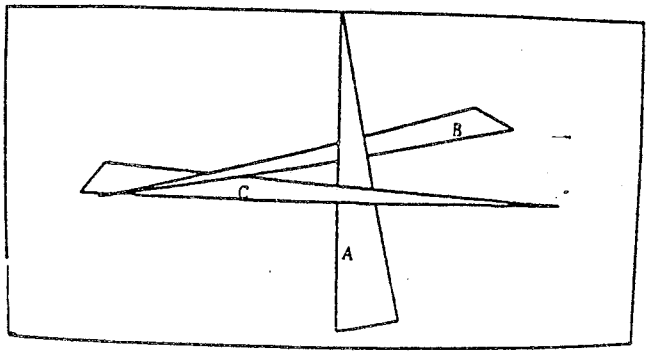


Figure 8: BSP tree and polygons of Fig. 6 (A and C have each been split into two parts by plane of polygon B)

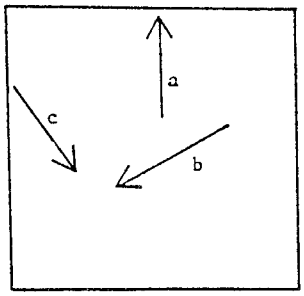


Figure 9a

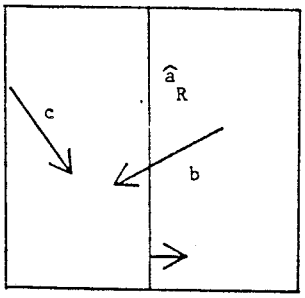


Figure 9b

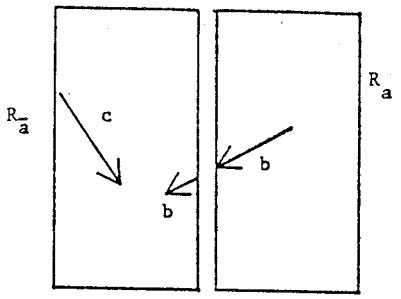


Figure 9c

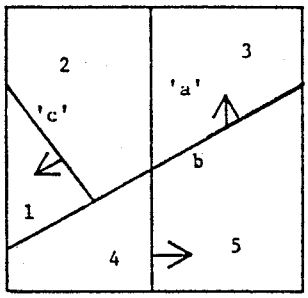


Figure 10

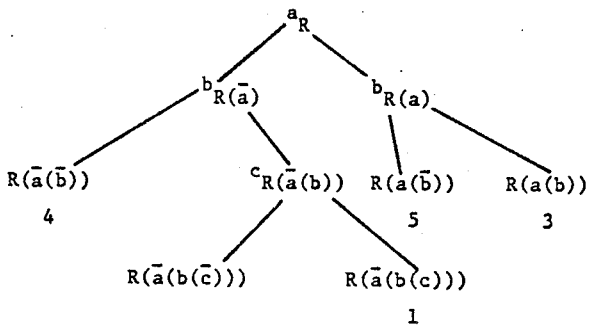


Figure 11

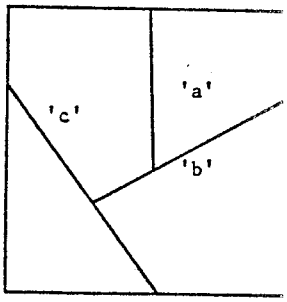
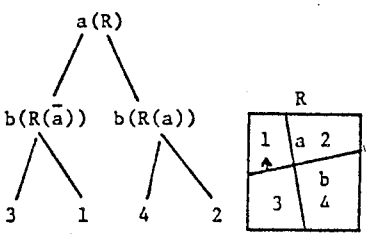


Figure 12



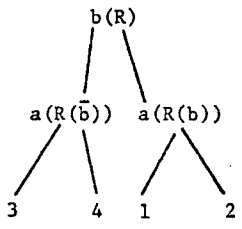
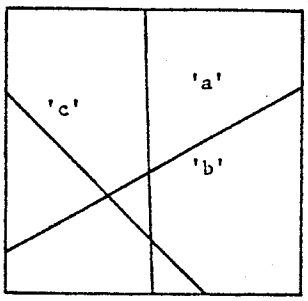


Figure 13



The maximum tessellation for Figure 9

Figure 14

(end)

ANNEXE 3 : UTILISATION DES COORDONNEES HOMOGENES

Cette technique de base permet de modéliser toute transformation géométrique par une matrice 3x3 ou 4x4, selon que l'on travaille dans un espace à 2 ou 3 dimensions.

L'emploi des coordonnées homogènes revient à représenter les objets de l'espace à N dimensions dans l'espace à N+1 dimensions, de telle façon qu'une projection perspective particuliére redonne l'espace à N dimensions. Ainsi un point de l'espace à 2 dimensions est représenté par le vecteur (x,y,w), le calcul pour retrouver le point initial étant le suivant :

$$\begin{cases} x2 = x / w \\ y2 = y / w \end{cases}$$

L'espace ainsi créé est formé de droites et de plans passant tous par l'origine, permettant ainsi la représentation en matrice 3x3. Il est à noter d'autre part que ces opérations matricielles sont facilement cablées.

Enoncons à présent les différents types de transformations en 3D, et les matrices asociées.

1. Translation

Soit Tx,Ty,Tz les composantes de la translation

$$T = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ Tx & Ty & Tz & 1 \end{vmatrix}$$

2. Rotation

Sont traités les trois types de rotations d'un angle t autour des axes Ox, Oy et Oz.

$$Rx(t) = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos.t & (-)\sin.t & 0 \\ 0 & \sin.t & \cos.t & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

$$Ry(t) = \begin{vmatrix} \cos.t & 0 & \sin.t & 0 \\ 0 & 1 & 0 & 0 \\ (-)\sin.t & 0 & \cos.t & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

$$Rz(t) = \begin{vmatrix} \cos.t & (-)\sin.t & 0 & 0 \\ \sin.t & \cos.t & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

3. Changement d'échelle (Zoom)

$$Z = \begin{vmatrix} Zx & 0 & 0 & 0 \\ 0 & Zy & 0 & 0 \\ 0 & 0 & Zz & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

4. Projection orthogonale

On cherche à présenter des vues frontales, de haut, et de coté.

Soit un point P(x,y,z,1)

$$(x,y,z,1) * \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} = (x,y,0,1)$$

Vue frontale : projection sur plan Ox,Oy

$$(x,y,z,1) * \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} = (x,z,0,1)$$

Vue de haut : rotation autour de Ox de pi/2
et projection sur Ox,Oy

$$(x,y,z,1) * \begin{vmatrix} 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} = (z,y,0,1)$$

Vue de coté : rotation autour de Oy de pi/2
et projection sur Ox,Oy

5. Projection perspective

C'est l'image définie par les intersections de rayons menés d'un point de vue C sur tous les points de l'objet, avec un plan orthogonal à une droite passant par C et appelée ligne de visée -cf.figure A3.1-.

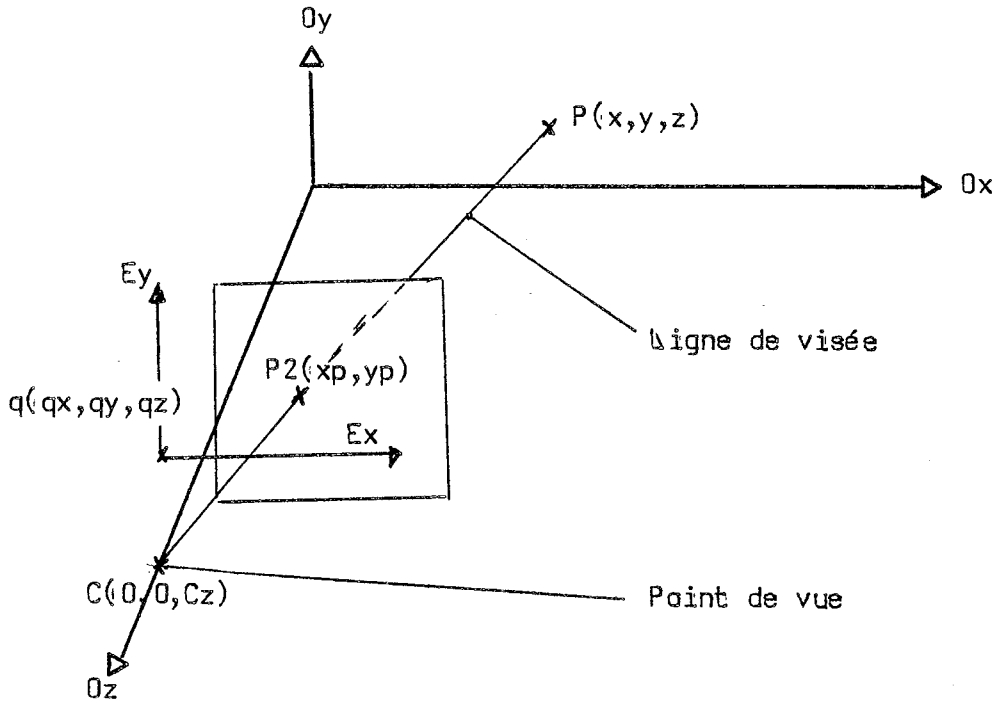


Figure A3.1: Transformation perspective

De ce schèma, se déduisent les relations suivantes :

$$xp = \frac{Cz - qz}{Cz - z} \cdot x \quad , \quad yp = \frac{Cz - qz}{Cz - z} \cdot y$$

La projection est supposée faite sur le plan z=0 (qz=0), d'où:

$$xp = \frac{1}{1 - \frac{z}{Cz}} \cdot x \quad , \quad yp = \frac{1}{1 - \frac{z}{Cz}} \cdot y$$

Ce qui nous amène à la matrice de projection suivante :

$$P = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1/Cz \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

ANNEXE 4 : MODELISATION DE L'ASPECT COULEUR DES OBJETS

1. Principes de la trichromie

Tous les procédés de reproduction de couleurs s'appuient sur la TRICHROMIE , principe énoncé par YOUNG, MAXWELL et HOLMHOLTZ, d'après lequel toute couleur peut être recréée par combinaison des 3 primaires Rouge, Vert et Bleu.

Ainsi toute couleur (C) peut être représentée à l'aide de 3 coefficients (r,v,b) indiquant l'intensité des primaires R,V,B:

$$m(C) = r(R) + v(V) + b(B)$$

m étant l'intensité de la couleur (C)

ce principe peut être illustré par la figure A4.1 (MAS81).

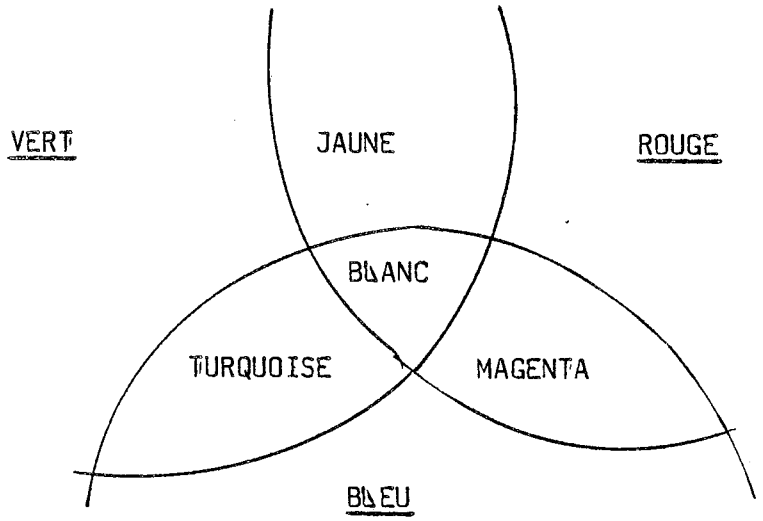


Figure A4.1: Principe de la trichromie

Cette formalisation présente l'inconvénient d'être peu naturelle pour un utilisateur "moyen". Elle fait appel en effet à la synthèse additive, moins familière que la synthèse soustractive utilisée en peinture et fonctionnant comme une série de filtres.

D'autres espaces peuvent être fournis à l'utilisateur :

- (Cyan, Magenta, Jaune) inverse de (R,V,B) dont les coefficients d'intensité sont c,m,j tels que:

$$(c,m,j) = (1,1,1) - (r,v,b)$$

- mais surtout (Teinte, Luminance, Saturation) objet du second paragraphe

2. Teinte, luminance et saturation

Ce sont ces trois paramètres prépondérants de la vision qui définissent l'espace de base.

La TEINTE associe la couleur à une notion de longueur d'onde déterminée. Dans le domaine du visible, cela va de 0.75 micron pour le rouge à 0.39 micron pour le violet.

La LUMINANCE caractérise l'intensité d'excitation ou de brillance, traduisant des couleurs plus ou moins lumineuses à éclairement égal.

La SATURATION indique le degré de pureté d'une couleur, distinguant couleurs vives et délavées.

On peut dire ainsi qu'une couleur est totalement définie par sa CHROMINANCE d'une part, association de teinte et saturation, sa luminance d'autre part.

L'espace (T,L,S) a été représenté par MUNSELL -cf.figure A4.2-.

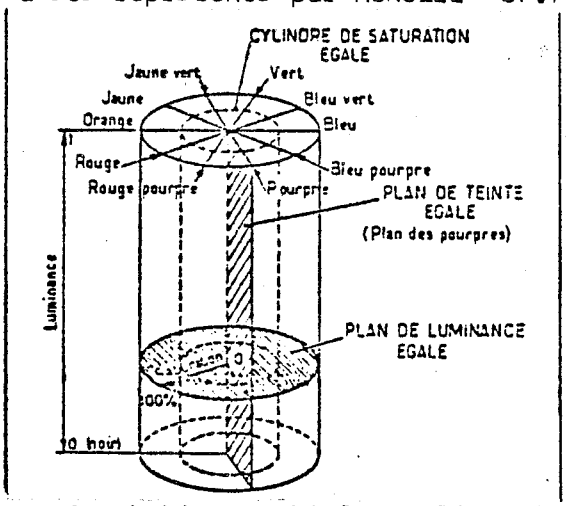
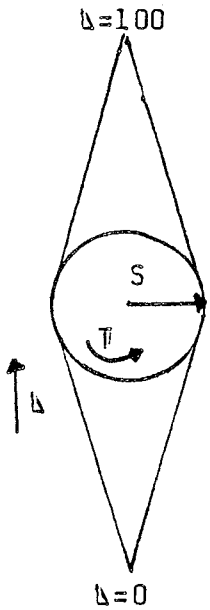


Figure A4.2: Représentation spatiale de l'ensemble [T,L,S]

Cet espace, facilitant le traitement numérique des informations de couleur par simple interpolation sur chacun de ses paramètres, a été repris par les grands constructeurs de terminaux graphiques comme en témoigne la figure A4.3 (MAS81).

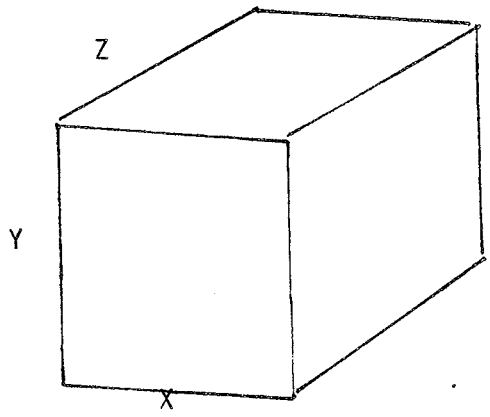


T varie de 0 à 360 degrés

S varie de 0 à 100
du centre vers la périphérie

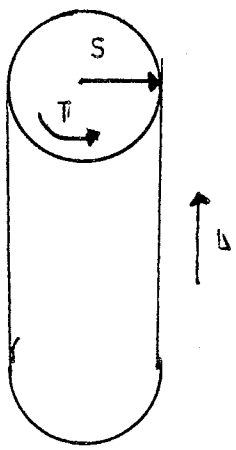
L varie de 0 à 100

Le double cone de TEKTRONIX



X = Rouge
Y = Vert
Z = Bleu

varient de 0 à 1



T,L,S varient de 0 à 1

Le cube et le cylindre de HEWLETT-PACKARD

Figure A4.3 : Utilisation des espaces (T,L,S) et (R,V,B)

ANNEXE 5 : EXEMPLE D'UTILISATION

Annexe 5

L'exemple présenté ici n'est qu'un très faible apercu des possibilités du logiciel réalisé. Nous nous devions toutefois d'illustrer, via un petit programme d'application et quelques dessins, les actions principales possibles et leur résultat.

Il est bien évident qu'une illustration plus étendue et plus démonstrative sera donnée lors de la soutenance de ce mémoire, cette fois à partir du terminal HELIOS.

L'exemple donné ci-après a été en effet traité sur le TEKTRONIX 4114, de par ses possibilités de "Hard-copy", permettant ainsi l'insertion de dessins dans ce document.

Ce programme d'application est constitué d'une part des déclarations en (EXTERNAL) des types d'attributs et primitives permis par le système -cf. module 'PRELUDE' cité en chapitre V- , d'autre part de l'application elle-même dont la finalité est la construction d'une structure graphique initiale, représentée par le premier dessin et que voici :

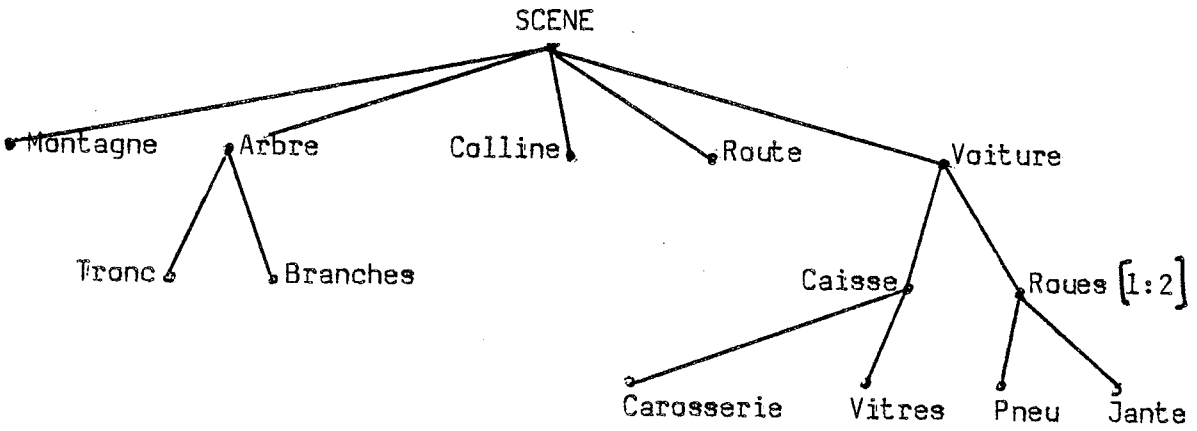
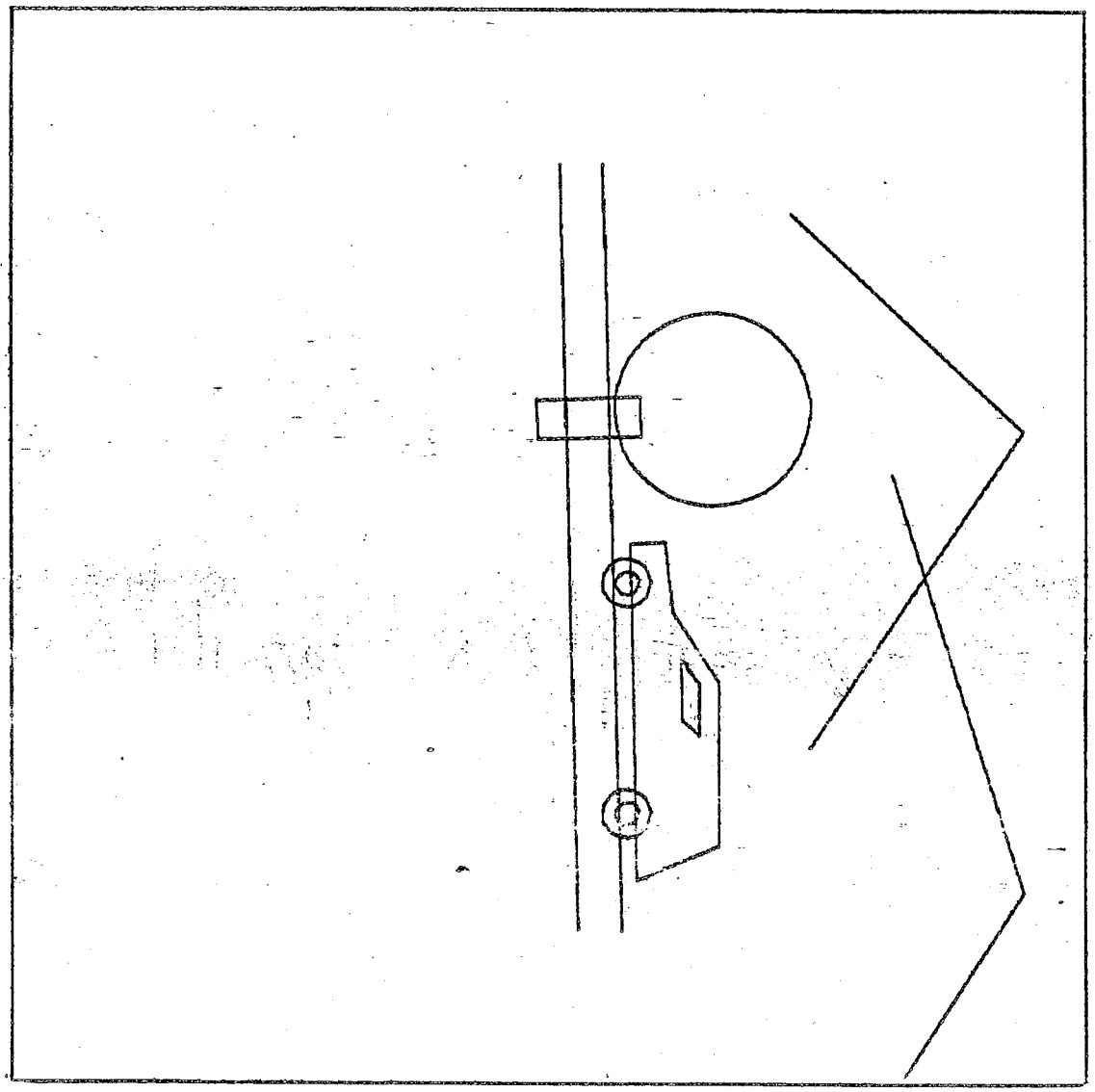


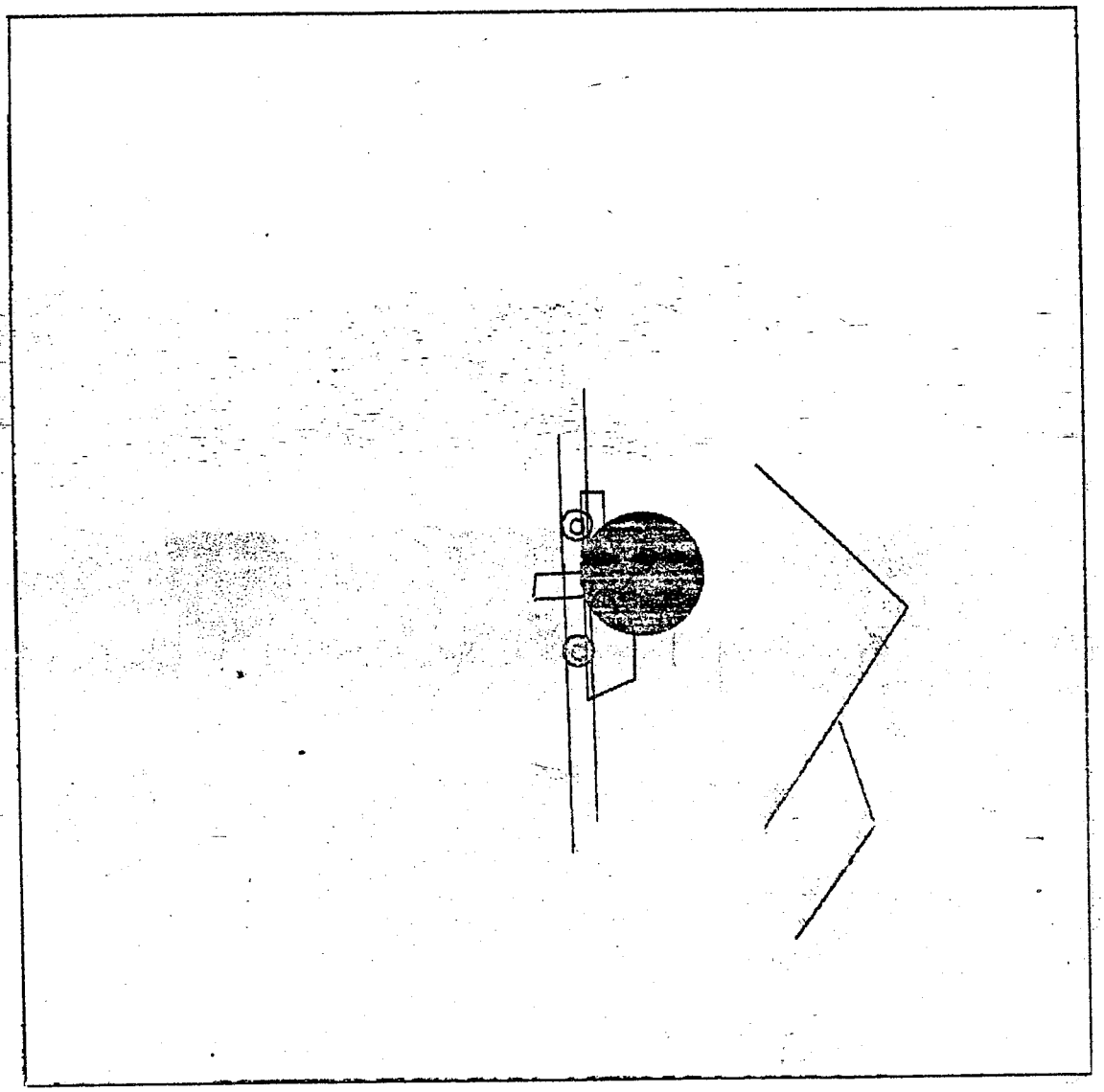
Figure A5.1: Représentation de la la scène présentée en exemple

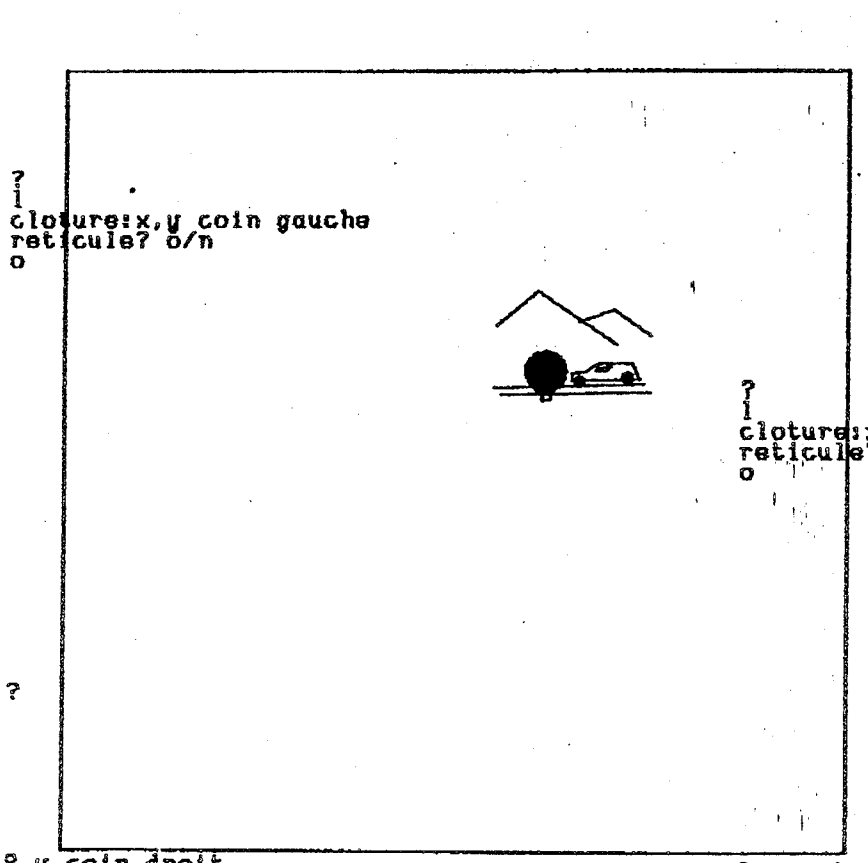
Deux phases sont à distinguer dans cet exemple :

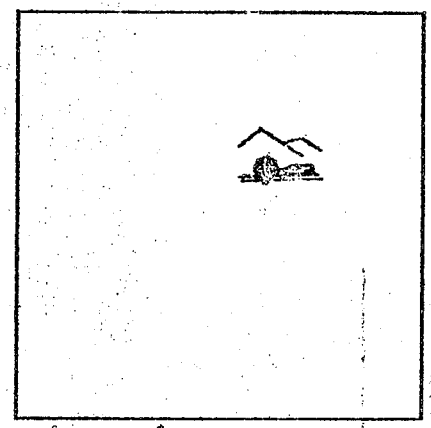
- éxécution du programme d'application, utilisant les principales primitives d'attribution morphologique -segments, ligne brisée, cercle-, de structuration -"STRUCTURES", "CONTEXTE", "IDENTITE"- et d'une primitive de description d'objets à partir d'une tablette à numériser TEKTRONIX -"POSITIONTABLETTE"-.

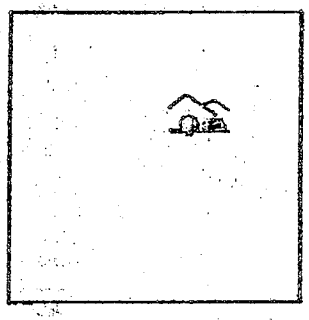
- actions interactives permettant notamment :
  * Zoom global de la scène -réduction-, remplissage de "branches" de "arbre", translation de "voiture", passant "derrière" "arbre".
  * Définition de plusieurs clotures représentant la scéne initiale
  * Nouvelle attribution morphologique et zoom local -grossissement- à "montagne", suppression de "arbre", création de "soleil" , ce affiché dans deux clotures.





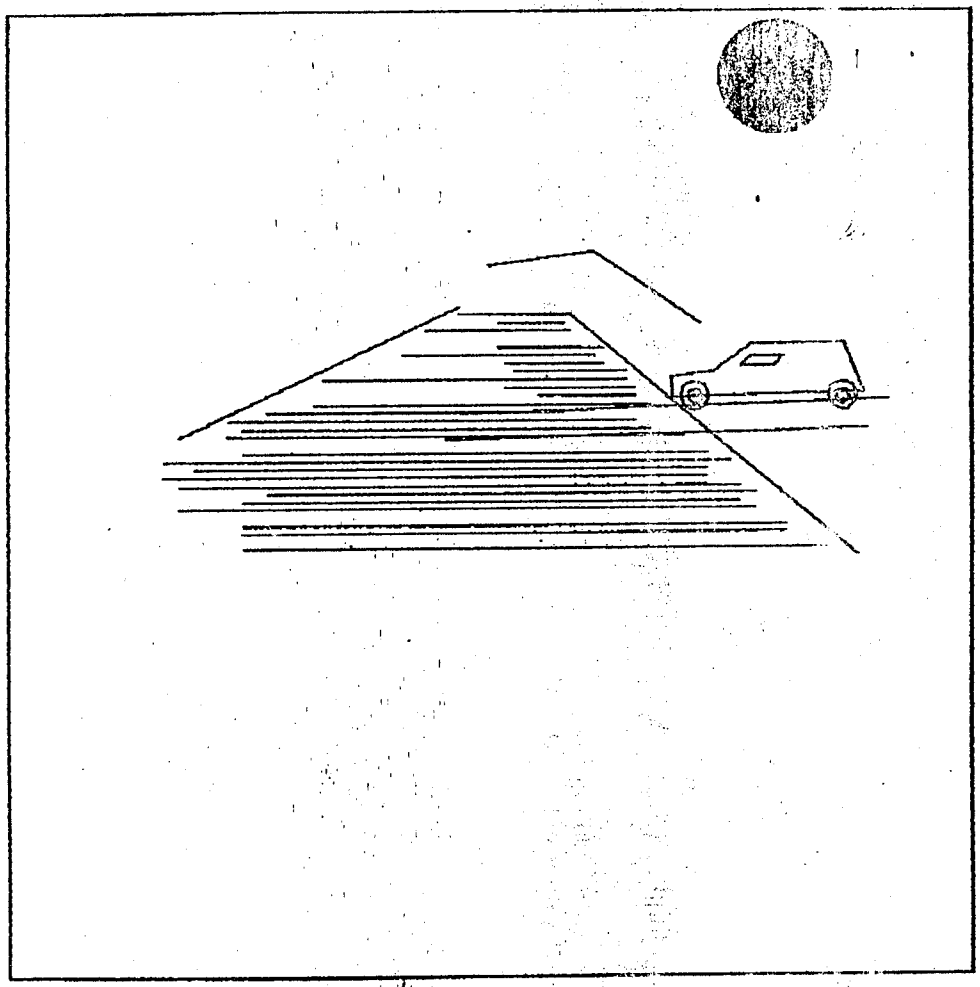

?
cloture:x,y coin gauche
reticule? o/n
o
?
x,y coin droit
v
absolue/relative [1,2]
1



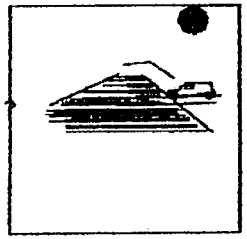


?
cloture:x,y coin gauche
reticule? o/n
o
x,y coin droit
v
absolue/relative [1,2]
1

```
?
x,y coin droit
absolue/relative [1,2]
1
cloture:x,y coin gauche
reticule? o/n
o
```



```
?
1
cloture:x,y coin gauche
reticule? o/n
o
```



```
module prelud(input,output);

const

(*codage du type de l'identite courante*)

(*--parametre 'identite'*)
      permanente=1;
      ephemere=2;

(*codage du type de visualisation demande*)

(*--parametre 'absolurelatif'*)
      absolu=1;
      relatif=2;

(*codage des attributs morphologiques de la classe des vecteurs2d*)

(*--parametre 'morpho'*)
      mpoints2d=74;
      msegments2d=75;
      mlignes2d=76;

(*codage des attributs morphologiques de la classe des cercles2d*)

(*--parametre 'morpho'*)
      mcercle2d=79;
      mdisque2d=80;
      marc2d=86;

(*codage des attributs morphologiques de la classe des texte2d*)

(*--parametre 'morpho'*)
      mtexte2d=72;

(*-------------------------------------------------------------------*)

(*codage des aspects*)

(* 1) pour la texture des segments*)

(*--parametre 'aspect'*)
      atrait_texture=112;

(*--parametre 'code'*)
      mixte=1;
      plein=2;
      tiretecourt=3;
      tiretelong=4;
      pointille=5;

(* 2) pour la taille des caracteres*)

(*--parametre 'aspect'*)
      acar_taille=101;

(*--parametre 'code'*)
      normal=6;
      taille2=7;
      taille3=8;
      taille4=9;


(*codage du parametre 'foe' dans CONSULTERVISUALISATION*)
```

```
        pfenetre=1;
        pcloture=2;
        pespaceutilisateur=3;

type
        typematrice2d3d=array[1..4,1..4] of real;
        tch=packed array[1..1] of char;
        ch80 =packed array [1..80] of char ;
        pliste=^listereels;
        listereels=record
                        x,y:real;
                        suiv:pliste;

                   end;
        typeidentite=integer;
        typeaspect=integer;
        typemorphologique=integer;

(*typedescripteur=mtexte2d..marc2d;
        enstypedescripteur=set of typedescripteur;*)

(*-------------------------------------------------------------------*)
        [external] procedure structures (pa : ch80);
        external ;
        [external] procedure attributvecteur2d (morpho : typemorphologiqu
            pt :pliste );
        external ;
        [external] procedure attributcercle2d (morpho:typemorphologique;p
            p3 : real);
        external ;
        [external] procedure attributextefdf (morpho:typemorphologique;x,
            y : real;lg:integer ;textes : ch80);
        external ;
        [external] procedure attributaspect (aspect,code : typeaspect);
        external ;
        [external] procedure rotation (xc,yc,angle : real);
        external ;
        [external] procedure translation (xd,yd,xf,yf : real);
        external ;
        [external] procedure zoom (K : real);
        external ;
        [external] procedure initialisergeometrie(mat1,mat2:typematrice2d
        external;
        [external] procedure identifier (var pr : boolean ;var ch : ch80)
        external ;
        [external] procedure fenetre(xg,yg,xd,yd:real);
        external;
        [external] procedure cloture(xg,yg,xd,yd:real);
        external;
        [external] procedure espaceutilisateur(xg,yg,xd,yd:real);
        external;
        [external] procedure positionecran (var x,y : real;var c:tch);
        external ;
        [external] procedure positiontablette (var x,y : real;var c:tch);
        external ;
        [external] procedure visualiser(absolurelatif:integer);
        external ;
        [external] procedure effacer(absolurelatif:integer);
        external ;

(*-------------------------------------------------------------------*)
```

```
                                                                    *)
    [external] procedure graphique ;
    external ;
    [external] procedure detruire ;
    external ;
    [external] procedure voirarbre ;
    external ;
    [external] procedure identitephemere(pa:ch80);
    external;
    [external] procedure identite (pa : ch80);
    external ;
    [external] procedure finidentite ;
    external ;
    [external] procedure contexte (ch : ch80);
    external ;
    [external] procedure fincontexte ;
    external ;

(*-------------------------------------------------------------------*)

(*-----------------------C O N S U L T A T I O N-------------------*)
    [external] function consulterstructure:ch80;
    external;

(*[external] function consultertype(typeattribut:enstypedescripteur):
        integer;
    external;*)
    [external] function consultervecteur2d:pliste;
    external;
    [external] procedure consultercercle2d(var p1,p2,p3:real);
    external;
    [external] procedure consultertextefdf(var x,y:real;var ls:integer;
        var textes:ch80);
    external;
    [external] procedure consulteraspect(aspect:integer;var code:integer);
    external;
    [external] procedure consultergeometrie(var mat1,mat2:typematrice2d3d);
    external;
    [external] procedure consultervisualisation(fce:integer;var xs,ys,xd,
        yd:real);
    external;
    [external] procedure consulteridentite(var identite:typeidentite;
        var pchaine:ch80);
    external;
    [external] function consultercontexte:ch80;
    external;

(*-------------------------------------------------------------------*)

(*---------------U T I L I T A I R E S-------------------------------*)
    [external] function lirentier (i : integer):integer ;
    external ;
    [external] function lireel (i : real):real ;
    external ;
    [external] function lirechaine (var chaine : ch80):integer;
    external ;

  [global] procedure scene;

  var
      i:integer;
      ct:ton;
      xt,yt:real;
```

```
        premierliste,precliste,ptrliste:pliste;

      procedure initnil;

      begin
        premierliste:=nil;
        precliste:=nil;
        ptrliste:=nil;
      end;

      procedure construireliste;

      begin
        if premierliste=nil
          then
            begin
              new(ptrliste);
              premierliste:=ptrliste;
            end
          else
            begin
              precliste:=ptrliste;
              new(ptrliste);
              precliste^.suiv:=ptrliste;
            end;
        with ptrliste^ do
          begin
            x:=xt;
            y:=yt;
            suiv:=nil;
          end;
      end;
  begin
    structures(
        'scene(arbre(tronc,branches))
    identite(
        'scene
    structures(
        'colline,montagne
    structures(
        'voiture(caisse(vitres,carosserie),roues[1:2](Jante,pneu))
    structures(
        'route
    finidentite;
    contexte(
        'scene(arbre)

(*construction de l'arbre*)
    writeln('TRONC');
    identite(
        'tronc
    initnil;
    for i:=1 to 5 do
      begin
        positiontablette(xt,yt,ct);
        construireliste;
      end;
    attributvecteur2d(mlignes2d,premierliste);
    finidentite;
    writeln('BRANCHES');
    identite(
        'branches
```

```
      initnil;
      positiontablette(xt,yt,ct);
      attributcercle2d(mdisque2d,xt,yt,100.0);
      finidentite;
      fincontexte;
      contexte(
          'scene(voiture(caisse))
      writeln('VITRES');

(*construction de la voiture*)
      identite(
          'vitres
      initnil;
      for i:=1 to 5 do
        begin
          positiontablette(xt,yt,ct);
          construireliste;
        end;
      attributvecteur2d(mlignes2d,premierliste);
      finidentite;
      writeln('CAROSSERIE');
      identite(
          'carosserie
      initnil;
      for i:=1 to 7 do
        begin
          positiontablette(xt,yt,ct);
          construireliste;
        end;
      attributvecteur2d(mlignes2d,premierliste);
      finidentite;
      fincontexte;
      contexte(
          'scene(voiture(roues[1:2]))
      writeln ('PNEU');

(*construction des roues[1:2]*)
      identite(
          'pneu
      initnil;
      positiontablette(xt,yt,ct);
      attributcercle2d(mcercle2d,xt,yt,25.0);
      finidentite;
      writeln('JANTE');
      identite(
          'jante
      initnil;
      positiontablette(xt,yt,ct);
      attributcercle2d(mcercle2d,xt,yt,12.0);
      finidentite;
      fincontexte;
      contexte(
          'scene

(*construction de la colline*)
      writeln('COLLINE');
      identite(
          'colline
      initnil;
      for i:=1 to 3 do
        begin
          positiontablette(xt,yt,ct);
          construireliste;
```

```
        end;
      attributvecteur2d(mlignes2d,premierliste);
      finidentite;

(*construction montagne*)
      writeln('MONTAGNE ');
      identite(
          'montagne
      initnil;
      for i:=1 to 2 do
        begin
          positiontablette(xt,yt,ct);
          construireliste;
        end;
      attributvecteur2d(mlignes2d,premierliste);
      finidentite;

(*construction route*)
      writeln ('ROUTE ');
      identite(
          'route
      initnil;
      for i:=1 to 4 do
        begin
          positiontablette(xt,yt,ct);
          construireliste;
        end;
      attributvecteur2d(msegments2d,premierliste);
      finidentite;
      fincontexte;
    end;

(*scene*)

end.
```

BIBLIOGRAPHIE

BOU83 A. BOUZE
Tendances et évolution de la synthèse d'image.
Probatoire CNAM - Grenoble - Mai 1983

BOU80 P. BOUILLE
Etude et réalisation d'algorithmes pour la visualisation de scènes composées de facettes planes.
Thèse Docteur-Ingénieur - Grenoble - Septembre 1980

BWJ84 W.F. BRONSVOORT, J.J. VAN WIJK, F.W. JANSEN
Two methods for improving efficiency of ray casting in solid modelling.
Computer Aided Design - Vol.16, no 1 - Janvier 1984

CHI83 K. CHIBANE
Interface spécialisée pour le synthétiseur d'images HELIOS.
Document interne au laboratoire ARTEMIS - Grenoble - Décembre 1983

CLA76 J.H. CLARK
Hierarchical models for visible surface algorithms.
ACM Communications - Vol.19, no 10 - Octobre 1976

EEK79 R. ECKERT, G. ENDERLE, K. KANSY, F.J. PRESTER
GKS79 : Proposal of a Standard for a Graphic Kernel System.
Congrès EUROGRAPHICS - Bologna - Septembre 1979

FER81 F. NUNES FERREIRA
Conception et réalisation d'un système interactif pour la synthèse d'images réalistes : HELIOS
Thèse Docteur-Ingénieur - Grenoble - Septembre 1981

FKN80 H. FUCHS, Z.M. KEDEM, B.F. NAYLOR
On visible surface generation by a priori tree structures.
ACM publication - Avril 1980

GEN84 P. GENOUD
Modélisation tridimensionnelle.
Document interne au laboratoire ARTEMIS - Grenoble - Janvier 1984

GRA84 J.F. GRABOWIECKI
Documentation système sur CLOVIS.
Document interne au laboratoire ARTEMIS - Grenoble - Mai 1984

GSP77 Status Report of the Graphics Standard Planning Commitee.
Computer Graphics - SIGGRAPH-ACM, Vol.11 no 2, 1977

HCY67 A. HURWITZ, J.P. CITRON, J.B. YEATON
GRAF : Graphic Additions to FORTRAN.
Spring Joint Computer Conference - 1977

HVI79 H. HVISTENDAHL
IGL : a structured language for Interactive Graphics.
Congrès EUROGRAPHICS - Bologna - Septembre 1979

LLM78 A. LEDUC-LEBALLEUR, M. LUCAS, F. MARTINEZ
Conception et réalisation d'un logiciel graphique interactif indépendant du contexte d'utilisation : le logiciel de base GRIGRI.
Revue RAIRO Informatique - Vol.12 no2 - 1978

LUC77 M. LUCAS
Contribution à l'étude des techniques de communication graphique avec un ordinateur. Eléments de base des logiciels graphiques interactifs.
Thèse d'Etat - Grenoble - Décembre 1977

MAF82 F. MARTINEZ, F.N. FERREIRA
HELIOS : Terminal vidéo interactif pour la synthèse d'images réalistes.
Le Nouvel Automatisme, no 30 - Mai 1982

MAR79 F. MARTINEZ
La synthèse d'images : revue bibliographique.
Convention LMT-MICADO - Grenoble - Janvier 1979

MAR82 F. MARTINEZ
Vers une approche systématique de la synthèse d'images.
Aspects logiciel et matériel.
Thèse d'Etat - Grenoble - Octobre 1982

MAR84 F. MARTINEZ
Vers une architecture banalisée des synthétiseurs d'images.
Premier colloque Image - Biarritz - Mai 1984

MAS81 C. MARSON
Terminaux : la couleur en plus
'01 Informatique' no 147 - Février 1984

MAT84 J.C. MARTY
Un éditeur graphique pour CASCADE : EDICAS
Thèse 3e cycle - Grenoble - Octobre 1984, à paraitre

MOL76 P. MORVAN, M. LUCAS
Images et ordinateur. Introduction à l'infographie interactive.
ed. LAROUSSE, série informatique - Septembre 1976

NES74 W.M. NEWMAN, R.F. SPROULL
An approach to Graphics System Design
Proceedings of the I.E.E.E., vol.62 no 4 - Avril 1974

SAR82 M.T. SARRASIN
Analyse et implémentation du logiciel pilote d'un synthétiseur d'images.
Mémoire d'Ingénieur CNAM - Grenoble - Octobre 1982

SET84 Special Images de Synthèse
Sciences et Techniques, numéro hors-série - Mai 1984

ZAR84 V.H. ZARATE
Architecture modulaire d'HELIOS.
Document interne au laboratoire ARTEMIS - Février 1984