THÈSE

présentée à

L'UNIVERSITÉ DES SCIENCES ET TECHNIQUES DE LILLE

pour obtenir

LE TITRE DE DOCTEUR INGÉNIEUR

par

Michel GRAVE

Sujet de la thèse:

ÉTUDE D'UN NOYAU DE SYSTÈME DE SYNTHÈSE D'IMAGES
APPLICATION À LA VISUALISATION DE SCÈNES TRIDIMENSIONNELLES

Soutenue le 15 décembre 1980 devant la Commission composée de:

MM.	**C.**	**CARREZ**	Président
	V.	**CORDONNIER**	Rapporteur
	M.	**LUCAS**	Examinateurs
	A.	**LEMAIRE**	

Je tiens à remercier

Monsieur le Professeur C. Carrez qui m'a fait l'honneur d'accepter de présider le jury,

Monsieur le Professeur V. Cordonnier qui a bien voulu se charger du rôle de directeur de thèse, et qui a fait preuve de beaucoup de patience au cours de ce travail.

Messieurs M. Lucas, professeur à l'université de Nantes, et A. Lemaire, ingénieur à l'ECMWF (Météo européenne à Reading (GB)) dont les conseils et les encouragements ont été particulièrement importants tout au long de l'élaboration et de la rédaction de cette thèse, ainsi que dans les années qui l'ont précédée.

Je remercie également tous ceux qui m'ont aidé au cours de ces dernières années, et plus particulièrement les membres du groupe graphique de l'INRIA.

La mise en page de ce texte a été réalisée à l'aide du système développé par l'équipe "communication homme-machine" de THOMSON-LCR.

Table des matières

1. INTRODUCTION

Cette thèse présente une étude dans le domaine de la production d'images à l'aide d'un ordinateur qui a été motivée par plusieurs constatations.

Depuis plusieurs années, on remarque que les terminaux destinés à la production d'images prennent une place de plus en plus importante en infographie. On constate cependant que les logiciels n'ont pas suivi cette évolution, et que les notions utilisées pour décrire des dessins sont encore employées pour décrire des images, ce qui limite énormément l'utilisation des richesses spécifiques à l'image. Ceci est particulièrement frappant lorsque l'on constate que les images obtenues à l'aide d'algorithmes récents ne peuvent pas être décrites avec les techniques traditionnelles.

Un certain nombre de logiciels existants ont certes été étendus de manière à permettre la description d'images, mais on remarque que la partie image et la partie dessin forment des parties disjointes dès qu'il s'agit d'autre chose que de modélisation géométrique.

Ces deux raisons ont donc motivé l'étude d'un système de synthèse d'image dégagé de toute idée de compatibilité avec les notions habituelles utilisées pour la production de dessins.

On remarque par ailleurs que l'évolution technologique actuelle tend à rendre très rapidement faux les arguments de coûts qui conditionnent les études en informatique depuis toujours, et que de plus les matériels sont de plus en plus adaptables. Les terminaux graphiques actuels sont en effet presque toujours construits avec des processeurs personnels programmables, et les problèmes de compatibilités deviennent alors moins cruciaux que par le passé. Ceci a donc conduit à la définition de notions indépendantes des terminaux pour la réalisation d'un noyau de système de synthèse d'images.

Le premier chapitre est consacré à la présentation des notions et du vocabulaire de base utilisés dans cette thèse. Les technologies et principes spécifiques à la synthèse d'images sont étudiés dans ce cadre.

Le second chapitre présente le noyau de système défini dans le cadre de cette étude. Ce noyau dénommé KISS(*1) a été défini de manière à permettre une mise en oeuvre aisée de diverses applications sans devoir réecrire à chaque fois les algorithmes élémentaires tels que tracé de vecteur, remplissage de facette ou découpage. De plus il a été conçu de manière a être aisément étendu. En particulier, les diverses notions présentées dans le chapitre précédent doivent pouvoir être utilisées sans nécessiter une modification du noyau. Ce second chapitre présente également quelques éléments pour la mise en oeuvre possible pour un tel noyau.

Le troisième chapitre montre comment une application peut être batie autour d'un tel noyau. L'exemple choisi est celui de la visualisation de scènes tridimensionnelles. Au cours de ce chapitre, des liens très étroits avec l'analyse de scènes sont présentés dans le cadre d'une utilisation d'un noyau tel que KISS.

Pour terminer, le quatrième chapitre indique quelques voies de recherche envisageables à partir des conclusions de cette étude, principalement dans le domaine de la parallélisation d'algorithmes et des structures fonctionnelles de matériels de synthèse d'images.

---

*1 Kernel for Image Synthesis Systems

II. NOTIONS DE BASE

1.Définitions

Une image est définie dans un plan par une fonction caractéristique (Shop74) qui sera notée a(x,y) et qui associe à tout point (x,y) un vecteur a d'un espace A appelé Espace des attributs. Nous supposerons dans un premier temps que A=Rn, n pouvant être quelconque.

En synthèse d'images, la fonction caractéristique est obtenue le plus souvent par combinaison de primitives, une primitive étant la fonction caractéristique d'un objet graphique élémentaire. Par exemple, la primitive droite(p1,p2) où p1=(x1,y1,a1) et p2=(x2,y2,a2) associera le vecteur attribut a = (1-t).a1 + t.a2 au point (x,y) = (1-t).(x1,y1) + t.(x2,y2) pour tout t, et le vecteur nul à tout autre point du plan.

La combinaison de deux primitives peut se définir de nombreuses manières, et nous verrons par la suite diverses définitions. A titre d'exemple, on peut citer la superposition, que l'on rencontre fréquemment, que nous noterons + et qui est définie de la manière suivante:

a1(x,y) + a2(x,y) = si a2(x,y) = 0 alors a1(x,y)
sinon a2(x,y)

On remarque que dans le cas général, la combinaison de deux primitives ainsi définie n'est pas commutative. Cette définition exprime la notion de priorité temporelle et on doit l'interpreter comme indiquant que la primitive 2 a été définie "après" la primitive 1.

Au cours du processus produisant une image il y a une étape d'échantillonnage de la fonction caractéristique, et de quantification de vecteur attribut (RoKa76). Le résultat de ces opérations est une image digitale qui est considérée comme étant un tableau rectangulaire dont les

éléments sont appelés pixels(*1). Un pixel est donc un vecteur attribut quantifié, mais nous utiliserons également ce terme pour désigner une portion élémentaire de la surface de visualisation.

Dans un système de synthèse d'images, on trouve principalement quatre grands composants. Le premier qui sera appelé synthétiseur produit des images digitales à partir de descriptions d'objets. Son rôle est donc de transformer les descriptions de scènes en listes de primitives, de combiner ces dernières et de discrétiser les fonctions caractéristiques. On constate que cet élément a à sa charge toutes les fonctions d'un système graphique classique. On trouve ensuite le convertisseur qui transcode les images digitales en informations nécessaires à l'apparition de l'image sur l'écran (*2) qui est le troisième élément. Le plus souvent, l'image digitale produite par le synthétiseur est mémorisée. C'est par exemple le cas lorsque le convertisseur doit raffraichir l'écran et que l'image ne peut pas être resynthétisée suffisamment rapidement. La mémorisation se fait alors dans une mémoire d'images qui est la quatrième composante importante d'un système.

On voit sur la figure 1 une représentation schématique de la structure d'un système de synthèse d'images. Les flèches représentent les circulations d'informations, et indiquent qu'en plus des communications par la mémoire d'images (facultative), le synthétiseur et le convertisseur peuvent avoir des échanges directs. Ils permettent principalement à ce dernier d'avoir des indications sur la manière d'interpréter l'image digitale. On note qu'il peut y avoir également des accès directs de l'application vers le convertisseur, ce qui n'apparait pas dans les systèmes graphiques traditionnels. Enfin, on remarque que ce que produit le convertisseur peut être à son tour rangé dans la mémoire d'images. Ceci impliquera le plus souvent que l'image source et l'image objet de la conversion soient dans des emplacements physiquement disjoints de la mémoire d'images.

La suite de ce chapitre est consacrée à une étude détaillée des diverses composantes et de leurs communications.

---

*1 Nous conserverons cette abréviation anglo-saxonne qui n'a pas d'équivalent français satisfaisant

*2 Ce terme sera utilisé pour désigner le support physique de l'image, même s'il s'agit de papier par exemple

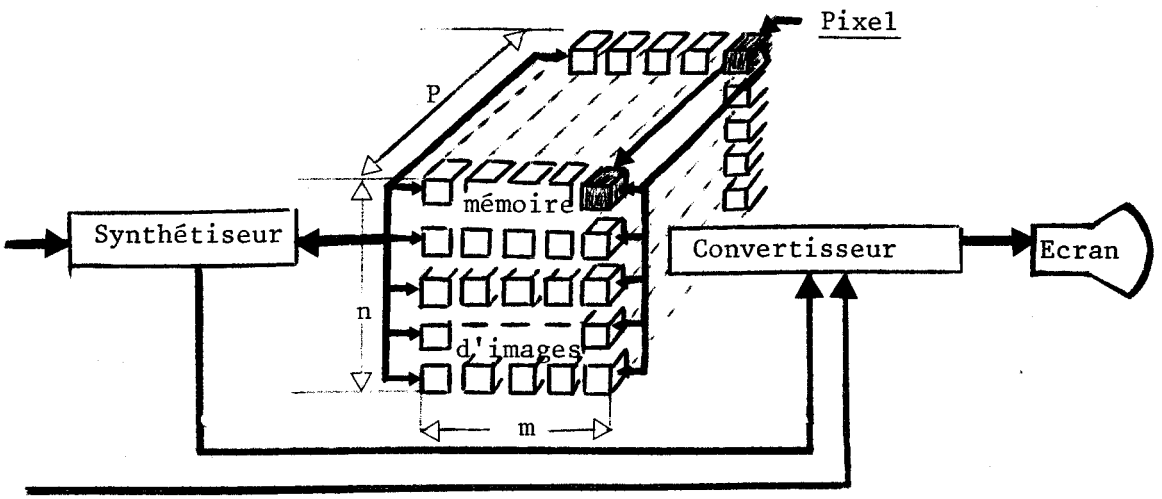


figure 1.

2.Ecrans et mémoires d'images

L'étude de ces éléments en premier lieu se justifie par le fait que leurs technologies ont une très grande importance sur les possibilités des systèmes, et sur les opérations que doivent réaliser les autres composantes. Nous débuterons avec une étude de quelques catégories d'écrans de manière à faire apparaitre leurs possibilités et leurs limitations. Nous en déduirons également les repercussions sur les mémoires d'images.

2.1.Ecrans à technologie vidéo

Les écrans qui ont le plus contribué au succès de la synthèse d'images sont sans conteste ceux qui utilisent les techniques vidéo pour produire des images. Les techniques employées sont identiques à celles qui sont utilisées en télévision domestique. Celà signifie que la surface de visualisation est balayée ligne par ligne par un (ou trois) faisceau(x), une image devant être raffraîchie 50 ou 60 fois par seconde suivant les cas (Cette notion de balayage ligne par ligne se retrouve de manière très

prononcée dans la plupart de algorithmes). Très souvent, le raffraichissement n'est effectué en réalité que 25 (ou 30) fois par seconde, en travaillant en mode entrelacé, ce qui signifie que pendant la moitié du temps de raffraichissement, on balaye les lignes de rang impair, et pendant la seconde moitié du temps, les lignes de rang pair. Ce procédé qui est tout à fait acceptable en télévision traditionnelle provoque parfois un scintillement de l'image en imagerie synthétique car dans ce cas, la cohérence entre deux lignes successives est moins forte.

Cette contrainte de raffraichissement est une des raisons pour lesquelles la résolution disponible avec les techniques video est limitée. On constate en effet que pour une trame m*n de pixels, le convertisseur doit calculer 25*m*n intensités par seconde (3 fois plus en couleur). On constate également le besoin de disposer d'une mémoire d'images pour éviter de synthétiser 25 images par seconde en synchronisme avec la conversion.

A quelques exceptions près, les terminaux que l'on trouve disposent de tubes couleur, ce qui offre beaucoup plus de possibilités pour la visualisation mais pose des limitations supplémentaires sur la résolution. La principale de ces limitations est due à la technologie des tubes. A l'heure actuelle, la visualisation d'images couleur sur trame de l'ordre de 500*500 est courante, on trouve également des tubes supportant un tramage 1000*1000, mais le cout en est plus élevé. En imagerie noir et blanc, on trouve parfois des trames 4000*3000 (*Prin).

2.2.Ecrans "bi-niveaux"

La deuxième catégorie regroupe les écrans que l'on qualifie de bi-niveaux à cause du fait que l'intensité d'un pixel n'a que deux valeurs possibles (en général allumé-éteint ou noir-blanc). On trouve principalement dans cette catégorie les imprimantes point à point (electrostatique (*Vers), à aiguille (*Prtr), thermique (*HP45)) ainsi que les écrans à plasma (*Orio, Slot76, Jarv76), et même dans certains cas des écrans à dessin (*TK14). Bien que basés sur des technologies très diverses, ces écrans présentent d'importantes propriétés communes.

La première est bien évidemment l'aspect bi-niveaux qui impose dans certains cas de mettre en oeuvre des méthodes de simulation de grisés, dont on trouve de nombreux exemples dans (JaJN76) ou (NeSp79). Une autre propriété commune importante est qu'ils ne nécessitent pas de

raffraîchissement d'image, ce qui enlève les limitations sur la trame (sauf dans le cas des plasmas) et les contraintes imposées à la mémoire d'images, qui n'est d'ailleurs plus toujours indispensable. Il faut également noter que dans la plupart des cas, l'image inscrite sur l'écran sera impossible à modifier.

2.3.Autres écrans

Il existe encore de nombreuses autres catégories d'écrans chacun ayant des particularités bien précises, mais un inventaire de ces différentes catégories n'entre pas dans le cadre de cette thèse. On peut cependant citer les plus fréquemment rencontrés, c'est-à-dire les C.O.M.,(*1) qui permettent d'impressionner directement les pellicules photographiques, et également les imprimantes traditionnelles qui ont été utilisées pour produire les premières images par ordinateur et qui le sont encore dans certains cas de mise au point.

3.Notions sur les synthétiseurs

3.1.Généralités

La définition du paragraphe 1 présente un synthétiseur comme un système graphique complet. Pour reprendre la terminologie introduite dans (Luca77), un synthétiseur doit donc assurer la préparation à la visualisation et les fonctions d'un système élémentaire habituellement appelé noyau de système graphique. La suite du présent chapitre est principalement destinée à présenter les divers éléments à prendre en compte pour la définition d'un tel noyau. Nous verrons de plus que tout le travail n'est plus nécessairement effectué par le synthétiseur, que certaines fonctions seront reportées au niveau du convertisseur, et que l'organisation d'un système de synthèse d'images est donc quelque peu différente de celle des systèmes graphiques traditionnels.

On trouve donc principalement quatre fonctions qui sont:

a). Application des transformations géométriques aux divers objets de la scène, et constitution d'une liste de primitives.

b). Découpage suivant une fenêtre de visualisation

---

*1 Computer Output Microfilm

c). Détermination des parties visibles (principalement dans le cas de scènes tridimensionnelles).

d). Production de l'image des parties visibles.

Dans de nombreux cas, ces diverses fonctions sont étroitement mêlées, au point qu'elles sont réalisées parallèlement par un algorithme unique qui les prend toutes en charge.

Nous allons dans ce qui suit examiner divers aspects de la fonction d) qui n'est pas la plus simple contrairement à ce que l'on constate dans le cas des systèmes de production de dessins. Nous verrons à cette occasion que le plus souvent, il ne sera pas souhaitable de dissocier les diverses fonctions d'un synthétiseur de la même manière qu'on le fait dans un système de production de dessins.

Dans ce paragraphe, nous supposerons qu'il existe une suite de primitives à combiner et à discrétiser. Pour simplifier, nous supposerons dans un premier temps que l'espace des attributs n'a qu'une dimension, l'attribut étant alors appelé intensité. De plus, les points à partir desquels seront définies les primitives auront des coordonnées exprimées dans un "repère écran" ce qui signifie que l'origine est situé au point bas gauche de l'écran, et que les valeurs entières de l'espace géométrique des primitives correspondront à des pixels de l'espace écran.

3.2.Echantillonnage et phénomènes d'aliasing

L'échantillonnage de primitives est une fonction qui a énormément d'importance en synthèse d'images. Le grand nombre de méthodes développées, ne serait-ce que pour produire des images digitales à partir de descriptions de traits est caractéristique de cette importance. Certaines d'entre elles ont d'ailleurs été présentées récemment (Cr78.1, BaFu79, PiWi80), ce qui montre que les solutions proposées à l'heure actuelle ne sont pas parfaites.

Dans l'ensemble des méthodes d'échantillonnage, on trouve principalement deux catégories de méthodes. La première regroupe celles qui échantillonnent chaque primitive séparément et combinent les résultats ensuite. La seconde regroupe celles qui combinent les primitives avant échantillonnage.

Les méthodes de la première catégories sont le plus souvent basées sur des algorithmes initialement conçus pour tables traçantes, et présentent, en contrepartie d'une grande simplicité de mise en oeuvre, quelques inconvénients liés pour la plupart au phénomène d'aliasing(*1) (RoKa76, Prat78). Ils se manifestent le plus souvent par l'apparition de "marches d'escalier" le long des traits, de pertes de détails et produisent souvent des figures de moiré qui perturbent l'image. La figure 2 montre un exemple de perturbations dues à ce phénomène.

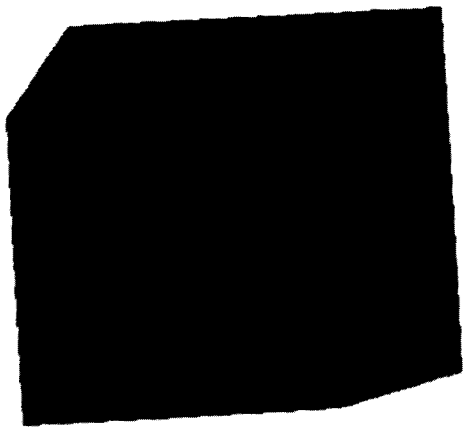


figure 2.

Sur cette figure on voit un cube sur les faces duquel des rayures ont été tracées. Géométriquement, ces rayures sont identiques sur les trois faces visibles, cependant, on remarque que leur apparence est très différente d'une face à l'autre.

Certaines améliorations peuvent être obtenues en procédant à un prétraitement des primitives, mais tous les problèmes ne sont pas résolus. C'est pourquoi les systèmes de synthèse d'images qui veulent produire une bonne qualité d'images utilisent pour la plupart des méthodes qui échantillonnent les primitives après combinaison. En réalité, c'est donc la

---

*1 Ce terme anglais difficilement traduisible désigne un problème qui découle du sous-échantillonnage d'une fonction. Il se produit de manière extrememement visible dans de nombreux cas où l'on échantillonne une image sur une trame de résolution insuffisante, sans précautions

fonction caractéristique de l'image qui est échantillonnée. Le plus souvent divers traitements tels que des filtrages seront appliqués à cette fonction avant échantillonnage, de manière à limiter les manifestations de l'aliasing.

On remarque que les deux catégories de méthodes sont très différentes. Dans le premier cas, chaque primitive "affecte" indépendamment des autres une valeur à chacun des pixels concernés, alors que dans le second cas chaque pixel "interroge" globalement la scène pour déterminer quelle valeur il doit prendre. L'utilisation d'une méthode de la seconde catégorie implique une mémorisation de toutes les primitives par le synthétiseur ainsi que la réception par celui-ci d'une indication de "fin de scène" qui n'est pas nécessaire pour les méthodes de la première catégorie.

Ces deux approches se retrouvent dans les systèmes de synthèse d'images traitant d'autres primitives, et principalement lorsqu'il faut procéder à une détermination de visibilité dans des scènes tridimensionnelles comme nous allons le voir dans le paragraphe qui suit.

3.3.Calculs de visibilité, problèmes de découpage

Lorsqu'il s'agit de produire une image à partir de la description d'une scène tridimensionnelle, on rencontre très rapidement le problème posé par la détermination des parties visibles (ou parties cachées) de la scène. En synthèse d'images, ce problème sera parfois étendu à des calculs de transparence, et de réflexions secondaires, mais les principes seront souvent identiques à ceux utilisés pour la visualisation d'objets mats et opaques. Nous verrons également que le problème de découpage suivant une fenêtre, que l'on rencontre même lorsqu'il est question de scènes bidimensionnelles, se pose de la même manière.

Dans les systèmes graphiques traditionnels, la détermination de visibilité a souvent été réalisée par des méthodes géométriques, le principe étant de transformer un ensemble de vecteurs en un autre ensemble de vecteurs. Les algorithmes de ce type s'appliquent difficilement en synthèse d'images et nous ne les aborderons pas ici (voir SSSc74).

Les autres méthodes se partagent en deux grandes catégories. La première regroupe les méthodes qui utilisent une approche primitive à primitive, la seconde étant composée des méthodes utilisant un approche globale.

Les méthodes de la première catégorie, qui utilisent des listes de priorité (NNSa72, NeSp79, SSSc74,..), consistent en la fabrication d'une liste ordonnée de primitives à partir de données géométriques dans l'espace de la scène, puis en un échantillonnage primitive par primitive, dans l'ordre indiqué par la liste. Lorsqu'une nouvelle primitive se trouve occuper des parties d'image où d'autres primitives avaient déjà une influence, il y a mélange des intensités, dans des proportions dépendant de l'opacité de la nouvelle primitive. Ce type d'approche pose bien évidemment les problèmes dus à l'aliasing que nous avons vus dans le paragraphe précédent, mais il pose également celui des découpages.

Dans certains cas en effet, il n'est pas possible d'ordonner l'ensemble des primitives tel qu'il se présente, et il devient alors nécessaire de procéder à un découpage dans l'espace de la scène (voir figure 3).

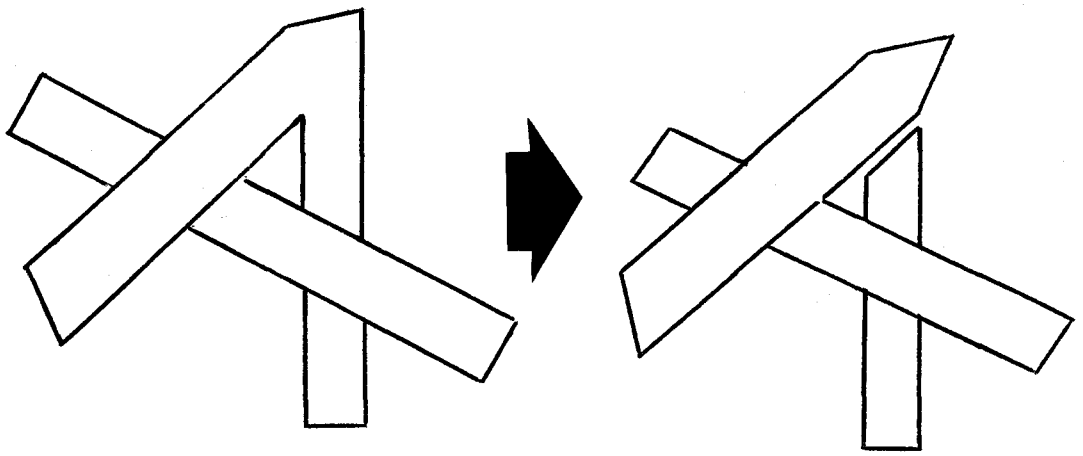


Figure 3.

Le découpage pose plusieurs problèmes que nous allons détailler.

Tout d'abord, il est bien connu que tous les objets graphiques ne se découpent pas selon un algorithme unique. L'exemple le plus célèbre est celui des facettes polygonales auxquelles il faut rajouter des arêtes fictives dans certains cas, alors que pour les courbes polygonales, ceci n'est pas nécessaire. On sait également que lorsque les contours ne sont pas polygonaux, les algorithmes habituels ne sont plus applicables.

Un problème plus important encore se pose lorsque l'on traite des primitives pour lesquelles les attributs ne sont plus uniformes. C'est le cas lorsque l'on veut représenter des ombrages ou des textures. En effet, il ne suffit plus de découper suivant les caractéristiques géométriques des

primitives, mais il faut également procéder à des découpages dans l'espace des attributs. Ceci n'est pas toujours possible, et il convient alors de procéder d'une autre manière.

Ces dernières années, plusieurs méthodes ont été décrites qui evitent les inconvénients liés au découpage et à l'aliasing dans les déterminations de visibilité. Il s'agit de méthodes dans lesquelles chaque pixel "interroge" la scène pour savoir quelle valeur il doit prendre. On retrouve donc à nouveau les techniques d'analyse globale de la scène en opposition aux techniques fonctionnant primitive par primitive.

La mise en oeuvre de ces techniques suppose principalement que l'on est capable d'inverser toutes les transformations geometriques qui ont été appliquées pour visualiser la scène, ainsi que pour décrire l'image que l'on veut en obtenir (perspective en 3D). Si cette condition est réalisée, alors chaque pixel est capable de retrouver la contribution de chaque objet élémentaire à la valeur qu'il doit prendre. Le problème de découpage peut également être éliminé avec des méthodes de cette catégorie, en remplaçant le découpage effectif d'un objet par une reduction de son domaine de visibilité comme il est suggéré dans (MaSh78).

Le principe général de cette approche consiste à associer à chaque primitive une fenêtre, qui subit les mêmes transformations que la primitive à laquelle elle est associée, mais qui pourra éventuellement être découpée puisqu'elle ne possède que des attributs géométriques. Lorsque toutes les transformations et découpages éventuels ont été effectués, on obtient une description de primitive accompagnée d'une description de surface plane dans l'espace de l'image, (la transformée de la fenêtre) permettant de ne prendre en compte que les parties visibles de la primitive à échantillonner. Lors de l'échantillonnage, un pixel ne tiendra compte de la primitive que si il est à l'intérieur de sa fenêtre, le découpage n'est donc effectivement réalisé que lors de l'échantillonnage. Lorsque l'un des objets de la scène doit être découpé en deux parties toutes deux visibles, il suffit de le dédoubler en associant à chaque copie une fenêtre différente.

Ce genre de technique est assez lié à l'utilisation de descriptions hiérarchiques de scènes, que nous allons étudier plus précisemment dans le paragraphe suivant.

3.4.Hierarchisations

Dans de nombreux domaines d'application, les scènes sont décrites d'une manière hiérarchique, ce qui se manifeste le plus souvent, de manière plus ou moins explicite par l'exploitation de structures arborescentes. Dans ces arbres, chaque feuille définit une primitive, et chaque branche porte des informations de découpage et une transformation géométrique.

Mallgren et Shaw (MaSh78) ont montré comment on pouvait remplacer l'exécution effective des découpages et transformations à chaque étape par une composition de ces entités jusqu'au moment de la visualisation. Dans le chapitre suivant, nous supposerons, par application de ces travaux, que toute définition de primitive est accompagnée d'informations pouvant résulter d'un tel cumul. On trouvera donc une transformation et un domaine (cumul de fenêtres) associés à chaque primitive. On remarque qu'un programme d'application ne doit à aucun moment exécuter des calculs de visibilité, mais qu'il doit se contenter de transmettre au noyau les informations qui lui seront nécessaires pour le faire.

Les descriptions hiérarchiques présentent de nombreux avantages, et parmi ceux-ci, la possibilité de décrire une scène d'une manière qui varie en fonction de la taille qu'aura sa représentation au moment de la visualisation (Clar76, RuWh80). Il est par exemple inutile d'utiliser la description des fenêtres d'un immeuble lorsqu'on sait que la taille de tout l'immeuble ne representera qu'un pixel sur l'écran. de même, si l'on associe à chaque noeud dans la hierarchie une enveloppe géométrique dans laquelle toute la descendance de ce noeud est incluse, il sera inutile d'exploiter plus loin la structure si l'on sait que cette enveloppe n'est pas visible. Lorsqu'on utilise ces techniques, on élimine les problèmes classiques tels que celui qui se pose lorsqu'on doit décider du nombre de segments à utiliser pour approcher un cercle sans connaitre la taille de celui-ci sur l'écran.

Ces techniques s'intègrent assez bien dans un système utilisant une méthode d'échantillonnage global, mais il se pose cependant le problème de déterminer, au niveau de la manipulation de pixels, comment cheminer dans la structure, et où s'arreter dans son exploitation. Nous verrons une solution à ce problème dans le prochain chapitre.

3.5.Primitives et attributs

La définition d'un ensemble aussi restreint que possible de primitives et d'attributs permettant de décrire les dessins fait l'objet de nombreuses discussions. Il est certain qu'il en sera de même pour les scènes devant permettre la production d'images.

Dans notre étude, nous avons tenu compte à la fois de ce qui a été écrit à propos des dessins (GSPC79, GKS79, Luca77,...), et d'une expérience vécue lors de l'étude d'une application particulière (GrOu79). Cependant, le fait de manipuler des images a eu une très grande importance et nous allons en voir certains aspects.

Le premier point important est celui qui consiste à décider si les primitives doivent être exprimées avec des coordonnées dans un espace tridimensionnel ou non. On peut remarquer qu'à ce sujet déjà, il existe des divergences. Le noyau présenté par le GSPC permet d'exprimer des coordonnées en 3D, alors que les systèmes présentés dans (GKS79) ou (Luca77) ne permettent que de travailler dans un plan.

En ce qui concerne la synthèse d'images, on note d'une part qu'une image (et une primitive au sens du paragraphe 1) est nécessairement définie dans un plan, et plusieurs expériences menées avant de définir un noyau nous ont montré que le plus souvent, la troisième coordonnée était manipulée comme un attribut. Ceci est particulièrement évident lorsqu'on étudie les algorithmes utilisant une mémoire de profondeur pour déterminer les parties visibles de scènes tridimensionnelles (Catm74, Park80). Ces algorithmes en effet font un échantillonnage suivant x et y et déterminent pour chaque couple (x,y) quel est son "attribut z", celui-ci étant ensuite rangé en mémoire d'image (sous conditions) de la même manière qu'une indication de couleur par exemple. Dans le noyau de système qui sera présenté dans le chapitre suivant, une position sera donc toujours dans un plan.

Un second point important est celui qui concerne le choix des attributs des primitives. Dans le cas qui nous interesse, ce problème sera reporté au niveau du convertisseur, et le synthétiseur considèrera un attribut uniquement comme étant une valeur sans signification, qu'il devra ou non ranger en mémoire d'image. L'interprétation de ces valeurs sera abordée dans le paragraphe traitant des fonctions d'un convertisseur.

Il reste enfin le problème posé par le choix des fonctions qui définissent une primitive à partir d'une suite de points. C'est-à-dire qu'il faut déterminer un ensemble de fonctions permettant de passer de la donnée de quelques points à la spécification des attributs (au sens donné à ce terme au début de ce chapitre) en tout point du plan.

Lors d'études précédentes (GrOu79, GGOu80), il a été décrit comment ce qui caractérise une primitive peut être séparé en plusieurs parties (voir également LeLM76). Dans le cas de la présente étude, une approche similaire a été retenue, dans un contexte simplifié. La séparation entre ensemble de points (géométrie dans GrOu79 ou sections dans LeLM76) et "méthode d'habillage" (resp. morphologie et mode) est conservée, mais chacun de ces éléments est un peu différent des définitions précédentes. Les points sur lesquels se définit une primitive sont des vecteurs dont les composantes ne sont plus seulement des coordonnées dans un espace géométrique, mais également des valeurs d'attributs. L'ensemble des points de définition d'une primitive sera appelé réseau. La "méthode d'habillage" que nous appellerons substance, consistera toujours à remplir l'intérieur de taches polygonales obtenues par interpolation linéaire sur les points du réseau. Nous justifierons ces choix par la suite, et principalement lors de l'étude de l'application à la visualisation de scènes tridimensionnelles.

3.6.Résumé

De ce qui précède, on peut retenir plusieurs choses:

- Tout d'abord, l'échantillonnage global est souvent souhaitable, et il est parfois indispensable. C'est le cas lorsque l'on ne dispose pas d'un support d'image digitale réutilisable (écrans bi-niveaux sans mémoire d'image intermédiaire). Un noyau de systèmes doit donc pouvoir être utilisé dans un tel contexte, ce qui implique entre autres l'existence de commandes explicites pour déclencher un échantillonnage (Debut-scène, Fin-scène).

- Certaines opérations telles que le découpage ne peuvent être exécutées qu'au dernier moment, pendant l'échantillonnage, et ceci implique que le noyau doit pouvoir être utilisé dans des approches où l'on utilise un cumul de fenêtres et de transformations.

- Le *mode* de travail par échantillonnage global impose aux pixels d'aller interroger la description de scène. Il faudra donc qu'un programme d'application fournisse au synthétiseur des outils lui permettant de venir l'interroger. Ceci se fera par exemple en fournissant au synthétiseur des procédures suivant un modèle prédéfini.

- Les espaces de coordonnées sont bi-dimensionnels, la troisième composante des vecteurs décrivant une position étant considérée comme attribut. Un programme d'application aura donc à sa charge les conversions éventuelles.

4.Fonctions d'un convertisseur

Un convertisseur doit produire un résultat sur un écran à partir de ce que lui fournit un synthétiseur, éventuellement par l'intermédiaire d'une mémoire d'image. Son role est assez mal défini en "graphiques", et est habituellement attribué aux "traitement d'images". Cependant, l'évolution technologique actuelle impose de plus en plus aux systèmes graphiques de tenir compte des fonctions de conversion, au sens qui est attribué à ce terme dans cette thèse.

Dans ce paragraphe nous nous bornerons à présenter séparément divers aspects de la conversion, en notant les implications matérielles qu'ils peuvent avoir. Nous constaterons à cette occasion les limitations (ou contraintes) importantes qu'impliquent les besoins de raffraichissement d'images (voir paragraphe 2) lorsqu'ils existent. Nous examinerons dans la fin de ce chapitre comment diverses fonctions peuvent coexister dans un système.

4.1.Tables de couleurs

Dans la plupart des systèmes produisant des images, il existe la possibilité d'utiliser ce que l'on appelle une table de couleurs et dont l'utilisation se fait suivant le principe schématisé sur la figure 4.

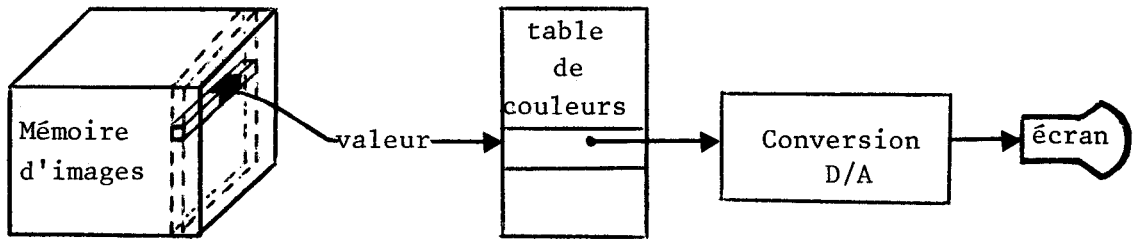


figure 4.

La valeur fournie par un attribut de pixel n'est pas utilisée directement pour calculer une intensité, mais sert d'index dans une table. A l'emplacement pointé par cet index, on trouve la valeur qui servira réellement au calcul d'intensité.

L'utilisation de ce dispositif est courante en traitement d'images et quelques publications ont montré que les tables de couleurs pouvaient jouer un role important en synthèse d'images (Sh79.2, SIBr79).

Nous distinguerons par la suite deux types de tables . Les premières qui seront appelées tables de compensation (Catm79) reçoivent une valeur d'index et fournissent en sortie une autre valeur entière. Les secondes, qui sont celles que nous appellerons tables de couleur fournissent trois composantes RVB pour chaque index. Nous verrons leurs rôles plus en détail dans le chapitre suivant.

Un des principaux intérêts de ces tables réside dans le fait qu'elles permettent de modifier très rapidement une image sans avoir à changer quoi que ce soit dans l'image digitale, c'est à dire sans faire intervenir un synthétiseur. Elles permettent également de limiter la taille des mémoires d'images (le nombre de bits/pixel) tout en conservant une grande gamme de couleurs disponibles.

4.2.Accès à la mémoire d'images

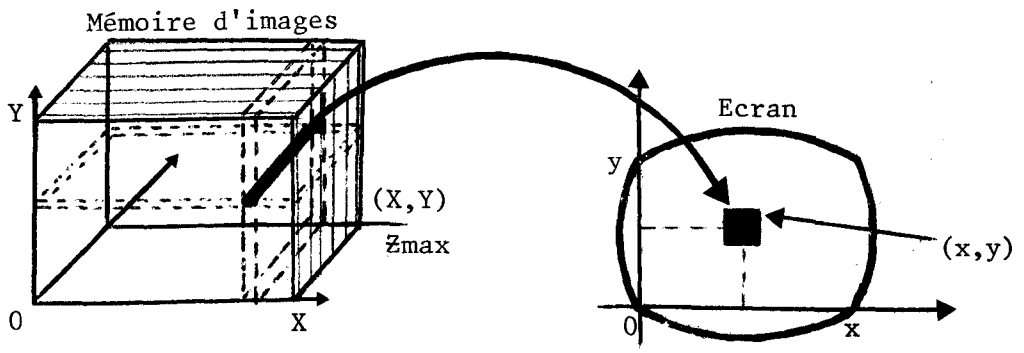


figure 5.

Prenons les notations indiquées sur la figure 5. Dans un convertisseur élémentaire, on aura toujours l'intensité en (x,y) sur l'écran calculée en prenant tous les bits de 0 à zmax du pixel (X,Y) de la mémoire d'image, avec x=X et y=Y. De plus en plus, l'association entre pixels de la mémoire d'image et pixels de l'écran se fait de manière moins directe. Le convertisseur devient alors capable de faire des opérations de découpage et des transformations géométriques entre l'espace (0,X,Y) de l'image digitale et l'espace (0,x,y) de l'écran. Ceci se manifeste souvent par la possibilité donnée à un utilisateur de déplacer une fenêtre dans l'image et de faire des effets de "loupe" sur cette image. Il faut noter que ces transformations deviennent de plus en plus complexes, principalement dans le domaine de la télévision où ces opérations sur des images digitalisées sont bien connues. Par ailleurs, un article de CATMULL et SMITH (CaSm80) montre une généralisation de ces manipulations(*1) ce qui tend à prouver que le rôle du convertisseur sera de plus en plus important dans l'avenir.

---

*1 Dans cet article, il est question de deux balayages pour transformer une image en une autre, ce qui suppose un stockage de l'image intermédiaire

4.3.Combinaisons d'images

Très souvent, il se révèle utile de combiner des images, c'est à dire de fabriquer une image à partir d'autres. De nombreux systèmes, dits de "traitement d'images", offrent de telles possibilités.

Il existe principalement deux grandes familles de combinaisons d'images, l'incrustation et le mélange.

Composer une image I3 par incrustation d'une image I2 dans une image I1 est une opération qui consiste à affecter à chaque pixel de I3 la valeur de son correspondant dans I1 ou de son correspondant dans I2 suivant un critère donné. En général la décision se fait suivant le schéma suivant:

si I2(x,y) = critère alors I3(x,y) := I1(x,y)
sinon I3(x,y) := I2(x,y)

On peut considérer que I3 est obtenue en mettant "I2 sur fond de I1".

Mélanger deux images I1 et I2 pour produire une image I3 est une opération qui peut se résumer comme suit:

I3(x,y) := (a*I1(x,y) op b*I2(x,y))/ c

Le plus souvent, op est un opérateur arithmétique etl que + ou -, et parfois * ou /, ou un opérateur logique, tel que et, ou,...
a, b et c sont 3 valeurs qui déterminent la proportion de I1 et I2 entrant dans I3, ainsi que l'"intensité" générale de I3. L'effet obtenu est semblable à ce que l'on obtient par superposition de deux transparents par exemple si l'opérateur est +. En Annexe 4, on trouve un exemple de convertisseur cablé permettant de réaliser ces fonctions.

4.4.Fonctions diverses

Il existe encore de nombreuses autres fonctions que peut assurer un convertisseur, mais nous terminerons avec deux d'entre elles uniquement. Tout d'abord, dans certains cas (Voir Annexe 4), l'image digitale peut être filtrée. Ceci est d'un usage courant en traitement d'images, mais l'est beaucoup moins en infographie (à ce stade de la fabrication de l'image). Nous ne nous en préoccuperons donc pas par la suite.

Une autre fonction importante peut être assurée par un convertisseur. Il s'agit de la simulation de niveaux d'intensités intermédiaires pour des écrans ne disposant que de peu de niveaux réels. Il existe de nombreuses techniques pour réaliser cette simulation (JaJN76, NeSp79), et nous ne les détaillerons pas. Ce type de fonction est particulièrement utile avec des écrans "bi-niveaux, et nous en verrons un exemple d'utilisation dans le chapitre prochain. Il faut noter que cette opération est souvent réalisée au niveau de la synthèse.

4.5.Conclusions sur la conversion

Ce paragraphe a montré la diversité des fonctions que peut assurer un convertisseur. Il faut remarquer que la mise en oeuvre de ces opérations dépend énormément de la configuration matérielle sur laquelle travaille le système de synthèse d'images.

On notera tout d'abord que lorsqu'il faut raffraîchir 512x512 pixels 25 fois par seconde, il faut que les traitements et accès mémoire se situant entre la lecture de mémoire d'image et la production de résultat visuel soient faits 6 millions de fois par seconde environ. Ceci implique nécessairement l'existence d'un processeur de conversion spécialisé utilisant des techniques de parallélisme dans les traitements (pipe-line par exemple). Un exemple en est donné dans l'annexe 4, et cet aspect des architectures de systèmes est traité dans le chapitre 4.

Cependant, de nombreuses applications sont développées dans des contextes ne nécessitant pas de raffraîchissement d'image (C.O.M, imprimantes électrostatiques), et on note également que le plus souvent, il est possible de construire une nouvelle image digitale à partir d'opérations sur d'autres, en utilisant des mémoires d'image supplémentaires.

Dans la suite de cette thèse, nous ne tiendrons donc pas compte des contraintes matérielles, considérant que tout peut être fait, à condition de disposer de mémoire ou de ne pas avoir de contrainte de raffraichissement.

5.Conclusions générales

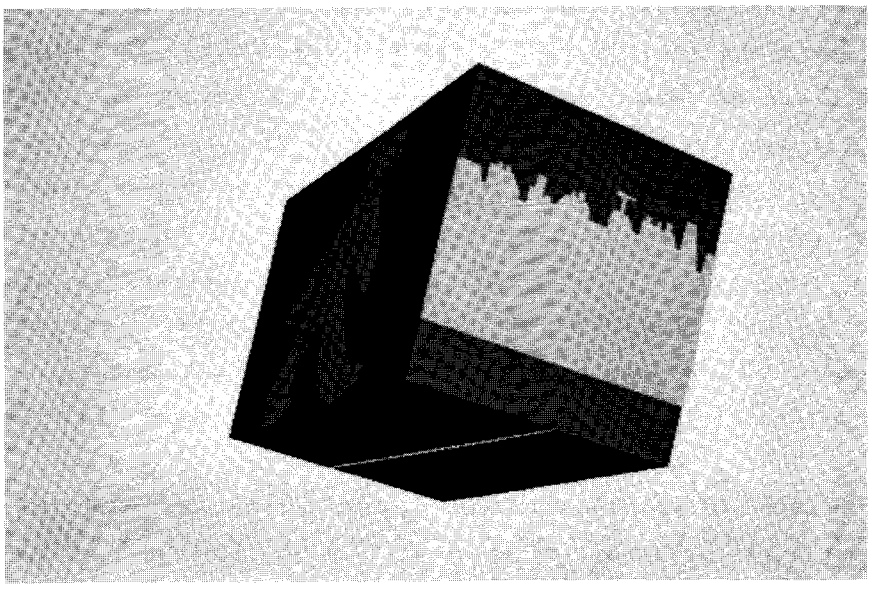
L'examen des possibilités offertes par les logiciels de description d'images existants (FoTD79, GKS79, LPSt80, Jone76,..) montre que le rôle attribué au convertisseur est pratiquement inexistant. En général, on ne fait qu'utiliser les possibilités d'accès à une table de couleurs. Par ailleurs, on constate qu'ils ne permettent le plus souvent que de définir des taches monochromes, le plus souvent uniformes. Un bon exemple de ce qu'offrent les logiciels en question est donné par le jeu de commandes permettant de décrire des images pour le TEKTRONIX 4027 (en partie décrit dans l'ANNEXE 2). Les commandes qu'il possède sont exactement les mêmes que celles qu'offrent les extensions de systèmes traditionnels(*1).

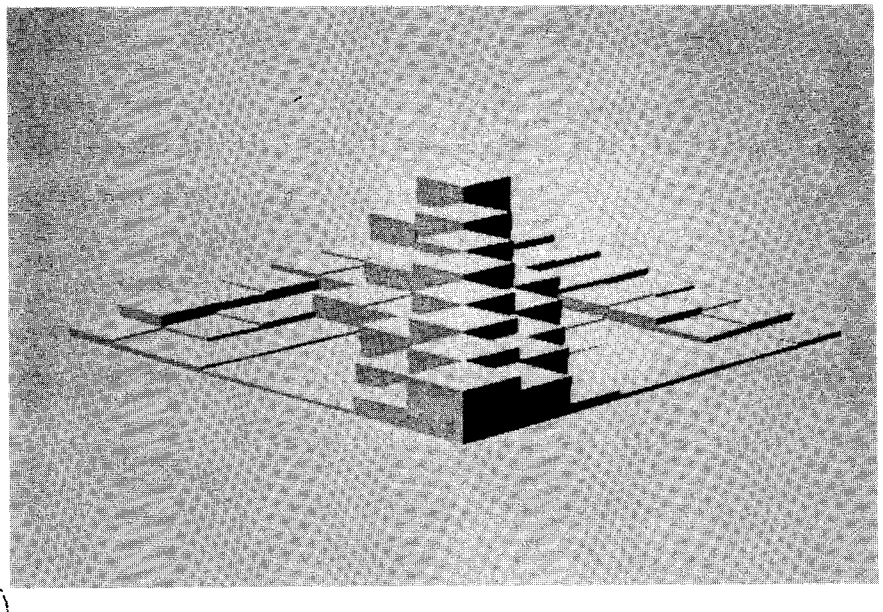
Dans de nombreux domaines d'application, un système graphique étendu de la sorte est suffisant. Les images de la figure 6 donnent quelques exemples de ce qui peut être obtenu sur un terminal tel que le TEKTRONIX 4027, à partir de programmes utilisant un logiciel traditionnel (en l'occurence, il s'agit de FORTRAN 3D, décrit dans (Salt79)). Ceci s'explique surtout par le fait qu'il n'est toujours principalement question que de descriptions géométriques d'objets, et ceci est parfaitement traité par les systèmes traditionnels si les précautions nécessaires sont prises lors des découpages.

Il reste cependant que, dès que l'on veut disposer de systèmes permettant la manipulation d'informations graphiques (et non plus seulement géométriques) riches, les problèmes évoqués au cours de ce chapitre se posent de manière assez importante, et de plus il est nécessaire de faire intervenir de nouvelles notions difficilement compatibles avec les structures traditionnelles des systèmes graphiques. Il est donc apparu souhaitable que la définition d'un noyau de système de synthèse d'images soit faite sans chercher à tout prix à rester compatible avec ce qui existe.

---

*1 Les commandes de description de textures sont cependant rarement utilisables à travers logiciels existants





Exemples d'images obtenues à l'aide de FORTRAN 3D

Figure 6.

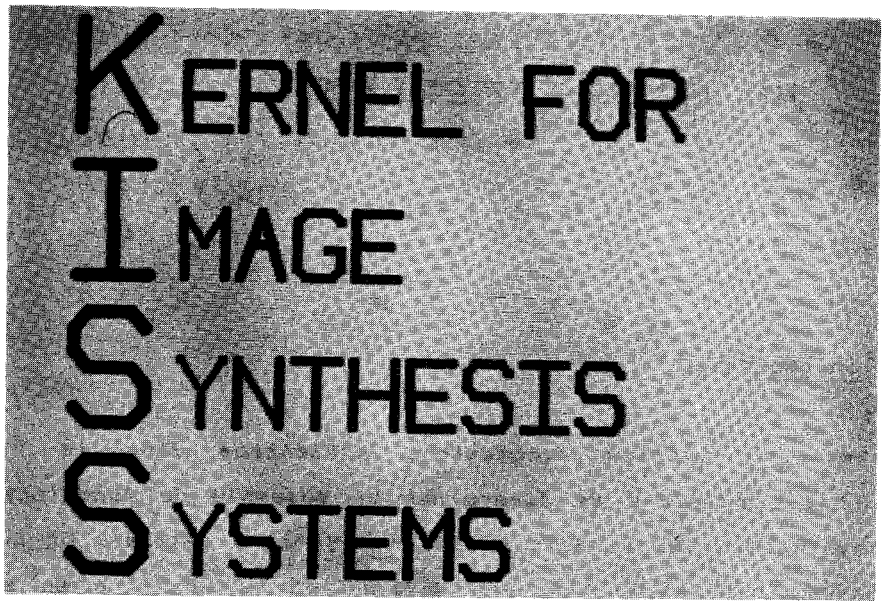


Figure 6.(suite)

III. LE NOYAU DE SYSTEME : KISS

1.Présentation générale

A partir des diverses conclusions présentées dans le chapitre précédent, il est possible de déterminer un ensemble de fonctions qu'un système de synthèse d'images doit posséder pour être général. Un noyau de système, appelé KISS(*1) a donc été défini, et ce chapitre en est une présentation.

KISS a été défini comme un noyau à partir duquel peuvent être construits des systèmes de synthèse d'images pour des domaines d'application très variés, ce qui imposait qu'il soit indépendant des contextes d'utilisation. De plus la grande diversité des matériels existants associée à leur grande souplesse de modification a conduit à une définition indépendante des matériels existants. Cependant, KISS ayant été spécifié en utilisant le langage PASCAL, cela implique que certaines caractéristiques spécifiques de ce langage ont été utilisées, et que certaines notions ont été définies en "termes PASCAL".

Toutes les fonctions de ce noyau ne peuvent pas être réalisées dans de bonnes conditions sur tous les matériels, et nous aborderons les problèmes de mise en oeuvre en fin de chapitre.

Un système utilisant KISS dispose principalement de quatre jeux de commandes. Le premier regroupe les commandes de gestion de la mémoire d'images et permet de décrire comment celle-ci doit être utilisée. Le second ensemble d'instructions destiné comme le premier au synthétiseur, permet de décrire les primitives qui vont être manipulées. Le troisième ensemble de commandes comporte celles qui sont destinées au convertisseur.

---

*1 Kernel for Image Synthesis Systems

Le quatrième ensemble contient toutes celles qui ne font pas partie des trois autres, et principalement des commandes de synchronisation ou d'attente d'evenements.

Nous allons tout d'abord examiner quelle machine de base est supposée supporter ce noyau, c'est-à-dire principalement quelle est sa structure supposée, et de quelles opérations élémentaires disposent les diverses composantes pour se communiquer des informations. Les divers ensembles d'instructions sont abordés dans les sections suivantes de ce chapitre.

2.La machine de base

Pour définir les divers éléments du noyau et leurs interactions, il a été nécessaire de définir une architecture de base que nous allons détailler dans ce qui suit. Cette machine se compose principalement d'une mémoire d'image, disposant de fonctions d'accès pour deux types de processeurs. Cette mémoire est organisée comme tableau tridimensionnel de taille (n*m*p), ces trois nombres pouvant être quelconques. Dans ce qui suit, nous noterons image(i,j,k) un élément (bit) de cette mémoire d'image, et nous appelerons point(i,j) l'ensemble (ordonné) de tous les éléments dont les deux premier indices sont i et j. L'ensemble de tous les éléments image(i,j,p) correspondant à une valeur particulière de p est appelé plan p de la mémoire d'images.

Un processeur accède à cette mémoire à l'aide de deux opérations élémentaires:

Lire(i,j,valeur,tranche)

et

Ecrire(i,j,valeur,tranche)

i,j et valeurs sont des nombres entiers, tranche est un couple d'entiers (t1,t2).

Lire affecte à valeur le contenu du mot composé des bits image(i,j,t1) à image(i,j,t2).

Ecrire affecte valeur au mot défini par i,j et tranche.

La notion de tranche qui apparait ici est particulièrement importante dans cette étude, et nous en verrons une utilisation dans le prochain chapitre. On peut noter que la valeur d'un pixel pourra être rangée dans une mémoire d'image, et que chaque attribut sera alors affecté à une des tranches qui auront été définies, d'une manière qui sera précisée dans la suite de ce chapitre ci. Par la suite, on appelera Tranche(t1,t2) de la mémoire d'images l'ensemble de tous les plans t1 à t2 de cette mémoire.

Pour implanter ce noyau, on pourra éventuellement modifier ces fonctions d'accès de manière à tenir compte de la physique des machines. On peut citer par exemple des ordres de mise à zéro de mémoire, d'écriture conditionnelle (Gr80.2), ou des commandes permettant d'accéder à des suites de pixels à l'aide d'une seule commande (BeBa80). Ceci sera ignoré par la suite afin de clarifier la présentation.

3.Primitives et commandes au synthétiseur

3.1.Définitions préliminaires

Il convient avant tout de définir quelques termes qui seront utilisés par la suite, ainsi que de préciser certaines conventions Tout d'abord, nous allons définir deux types.

```
type Rectangle = struct
                    xmin : entier ;
                    ymin : entier ;
                    xmax : entier ;
                    ymax : entier ;
                     fin

type Tranche = struct
                  t1 : entier ;
                  t2 : entier ;
                  v1 : entier ;
                  v2 : entier ;
             suivant : &Tranche ;(*1)
                   fin
```

---

*1 Pour des raisons typographiques, le symbole & est utilisé pour noter un pointeur au sens PASCAL

Une image digitale sera rangée dans une mémoire d'image d'une manière qui sera décrite dans un objet de type Trame dont la définition est la suivante:

```
type Trame = struct
             F : Rectangle ;
             C : Rectangle ;
    ntranches : entier ;
             T : &Tranche
              fin
```

Les éléments F et C d'une Trame sont utilisés comme définitions de fenêtre et cloture (LeLM76) et indiquent donc comment doit se faire la correspondance entre éléments dans l'espace des primitives et éléments dans l'espace de la mémoire d'image. On notera que tout se décrit à l'aide de valeurs entières.

Il n'existe pas à priori d'espace normalisé pour les coordonnées qui seront fournies au noyau de système, et les coordonnées seront exprimées par des nombres entiers. Le programme d'application appelant aura donc à sa charge les conversions éventuelles.

La liste de tranches associée à une Trame jouera un rôle qui sera précisé dans le paragraphe suivant.

3.2.Définitions de trame

Nous allons tout d'abord examiner les commandes permettant de décrire l'utilisation de la mémoire d'image, c'est à dire comment sont organisées les tranches, et quelles parties de la mémoire d'image seront utilisées. C'est à partir de ces commandes que seront spécifiés des objets de type Trame.

def-fenêtre (fxmin,fymin,fxmax,fymax: entier; T: Trame)

Cette commande permet de définir un rectangle dans l'espace des primitives et est utilisé pour déterminer des facteurs d'echelle et une translation en conjonction avec une definition de cloture. T est le nom de la trame concernée par cette définition, c'est à dire celle dont on va initialiser le champs F.

def-cloture (cxmin,cymin,cxmax,cymax: entier; T: Trame)

Cette commande, complémentaire de la précédente, définit un rectangle dans la mémoire d'image qui sera appelé partie active de la mémoire d'image.

def-tranche (n,t1,t2,v1,v2: entier; T: Trame)

Par cette commande on indique que la tranche référencée ultérieurement par le numéro n, dans la trame T, est constituée des plans t1 à t2 (inclus) de la mémoire d'image, et que si l'on affecte la valeur v à un élément de cette tranche, il faut inscrire:

```
si  v ≤ v1  alors  0
            sinon si  v ≥ v2  alors  Max-1
                              sinon entier(Max*(v-v1)/(v2-v1))
```

Avec: Max = 2**(t2-t1+1)

Lors de la définition d'une tranche de numéro n, toutes les tranches de numéro 1 à n-1 qui n'existent pas sont définies "vides". Cette commande peut être utilisée pour redéfinir des tranches existantes.

On remarque qu'une telle commande permet, sans autre modification que celle des paramètres t1 et t2 de faire exécuter le même algorithme sur des tranches de caractéristiques diverses, permettant ainsi à un programme de s'adapter à la configuration sur laquelle il s'exécute. Les rôles de t1, t2, v1 et v2 sont assez semblables à ceux des valeurs définissant les fenêtres et clotures que nous avons vus précédemment.

ini-tranche(n,v: entier)

Cette commande annexe initialise la tranche n (définie lors d'un appel préalable de def-tranche) en affectant à tout élément de la tranche la valeur v.

3.3.Primitives et attributs

Lorsqu'une fenêtre, une clôture et des tranches ont été définies, il est possible de donner des ordres de remplissage dans les tranches. Ceci se commande tout d'abord en décrivant des primitives puis en donnant des ordres d'echantillonnage de ces primitives.

Le type Primitive se définit de la manière suivante:

```
type Primitive = struct
                 n : entier ;
                 D : Domaine ;
                 T : Transformation ;
                 R : &ListePoints ;
                 S : &Substance ;
                  fin
```

Nous étudierons chacun des divers champs dans le cours de ce paragraphe.

Notons tout d'abord la commande suivante:

Associer (n: entier; D: Domaine; T: Transformation; R: ListePoints; S: Substance)

Cette commande indique au synthétiseur que la primitive qui sera nommée n sera composée du Domaine D, de la Transformation T, de la liste de points R, et de la substance S. Suivant la manière dont le synthétiseur est mis en oeuvre, cette primitive sera echantillonnée des qu'elle sera définie ou rangée dans une liste qui sera traitée sur demande explicite de l'utilisateur. Cette demande se fait par l'ordre:

Fin-Scene(n: entier)

Une telle commande est inopérante dans le cas d'un échantillonnage primitive à primitive, mais est indispensable dans le cas d'échantillonnage global. n est un paramètre dont le rôle sera précisé lors de l'étude de la commande de synchronisation.

Pour permettre d'initialiser les données utilisées pour stocker temporairement les primitives, il existe également un ordre:

Debut-Scene(T: Trame)

T indique la Trame qui définit l'image digitale que l'on désire manipuler. Cet ordre ne fait pas d'initialisation d'image digitale. Lorsque deux ordres Debut-Scene se succèdent sans avoir été séparés par un Fin-Scene, on considère que tout se fera avec la nouvelle trame, sans plus se préoccuper de la précédente (il n'y a pas de génération de Fin-Scene implicite).

Un point est défini par ses coordonnées (x,y) et son vecteur attribut

a, et un réseau est une liste de points. Pour construire un réseau, il existe deux ordres:

ini-reseau(nom,ndim:entier)

Cette commande initialise un réseau qui sera ultérieurement reperé par nom, et dont chacun des points aura ndim composantes (x,y et ndim-2 attributs). Après exécution de cet ordre, le réseau concerné est vide.

ajouter-point(nom: entier; element: point)

Par cette commande, on ajoute un nouveau point au réseau qui est repéré par nom.

Le type Point se définit comme suit:

```
type Point = struct
             Dim : entier ;
             X : entier ;
             Y : entier ;
             A : &Attributs;
              fin
```

Implicitement, l'attribut i d'un point à écrire en mémoire d'image, ou à lire depuis celle-ci, est associé à la tranche i de la trame courante. Lorsqu'un attribut est associé à une tranche vide, l'accès est ignoré pour cet attribut, mais pas nécessairement pour tous les autres.

Un objet de type substance est une procédure qui admet en entrée un point et qui produit en retour un autre point. Nous verrons son rôle plus en détail par la suite, mais nous pouvons noter pour le moment que ce sera par une définition de substance que l'on décrira des textures par exemple.

Un objet de type domaine sera une liste de points dans le cas général, mais sera souvent ramené à un objet de type Rectangle pour alléger les traitement de synthèse. Comme ce type sera dépendant des mises en oeuvre, nous ne le définirons pas plus précisemment dans ce chapitre. Il en sera de même du type transformation.

3.4.Exemple de procédure de synthèse

Avant de continuer la présentation des commandes permettant à un programme d'application de décrire une image digitale, nous allons étudier la structure fonctionnelle d'un noyau sur un exemple simplifié de noyau PASCAL. Ceci permettra de préciser divers types qui seront utilisés par la suite, ainsi que le rôle de certaines procédures.

Nous allons décrire des procédures PASCAL qui traduisent un ensemble de primitives en valeurs d'attributs pour chaque point d'une image digitale. Il s'agit d'une méthode proche de celle décrite par Warnock (Warn69) et qui utilise donc une méthode globale d'échantillonnage.

```
type PointElement = struct
                    P : point ;
                    Suivant : &PointElement ;
                     fin

type ListePoints = struct
                   Taille : Entier ;
                   Tete   : &PointElement;
                    fin

type PrimitiveElement = struct
                        P : Primitive ;
                        suivant : &PrimitiveElement ;
                         fin

type ListePrimitive = struct
                      Taille : entier ;
                      Tete   : &PrimitiveElement ;
                       fin
```

```
procedure Synthetiser( Z:Domaine ; Scene:ListePrimitive ; T:trame);

var z1,z2,z3,z4 : Domaine;
var prim : Primitive;
var Sc2 : ListePrimitive;
var i : entier;

debut
Sc2.taille := 0;
pour i := 1 a Scene.Taille faire debut
            prendreprimitive(prim, scene);
            intersection (var prim&.D, Z);
             si nonvide(prim&.D)  alors
                                   ajouter(prim,Sc2);
                                  fin

si Sc2.Taille <> 0 alors debut
                  si Test (Z) alors resoudre(Z,Sc2,T);
                                 sinon debut
                          . . . . . .
            faire de z1,z2,z3,z4 une partition de Z
                          . . . . . .
                              Synthetiser(z1,Sc2,T);
                              Synthetiser(z2,Sc2,T);
                              Synthetiser(z3,Sc2,T);
                              Synthetiser(z4,Sc2,T);
                                    fin
                         fin
fin
```

Plusieurs procédures sont utilisées dans ce qui vient d'être défini, mais nous ne nous intéresserons qu'à intersection, test et Resoudre, les autres relevant plutôt de la gestion de listes et n'étant pas du ressort de notre étude.

a)- intersection(var D : Domaine ; Z : Domaine) est une procédure qui retourne dans D l'intersection des deux Domaines D et Z. Dans le cas général, un objet de type Domaine est constitué d'une liste de couples (x,y) et d'une valeur indiquant le nombre de couples.

b)- test(Z : Domaine) est une fonction qui prend la valeur vrai ou faux suivant le resultat des opérations sur Z. En général il s'agit de comparer la surface de Z à une valeur prédéfinie.

c)- Resoudre(Z: Domaine ; Scene: ListePrimitive ; T: trame) est une procédure qui peut varier d'un type d'application à un autre. Son rôle est de déterminer la valeur à donner aux attributs de tous les pixels de Z, sachant que cette valeur est constante sur Z. Il lui faut donc déterminer pour chaque primitive de Scene quelle est sa contribution dans Z. Ceci est obtenu en déterminant quelle partie de Z est couverte par la primitive, et quelle est la valeur moyenne de la primitive sur cette partie. Ensuite, les divers résultats obtenus sont combinés d'une manière qui variera d'une application à l'autre.

Dans une procédure Resoudre, on trouve un appel qui est de la forme suivante:

Integrer(Z , primitive , point)

Après exécution de cette procédure, point contient ce qui sera considéré comme la valeur moyenne des attributs de primitive sur Z. L'existence de cette procédure est particulièrement importante, et nous l'aborderons en détail par la suite.

On trouve également dans Resoudre, deux appels à des procédures devant permettre la lecture et l'ecriture de pixels. Ce sont:

Prendre(point , trame)

et

Mettre(point , trame)

Ces deux procédures vont respectivement lire et ecrire un point dans la mémoire d'images de la manière dont trame l'indique. Elles jouent un rôle très important en ce qui concerne l'adaptation du noyau à une configuration matérielle donnée, puisque ce sont, du point de vue du synthétiseur, les seules qui connaissent le support physique de l'image digitale. Pour travailler, ces deux sous-programmes devront utiliser une description d'associations entre attributs et tranches.

3.5.Procédures utilisateur

De ce qui précède, nous pouvons déduire une structure de programmes produisant une image digitale. La figure 7 représente cette structure.

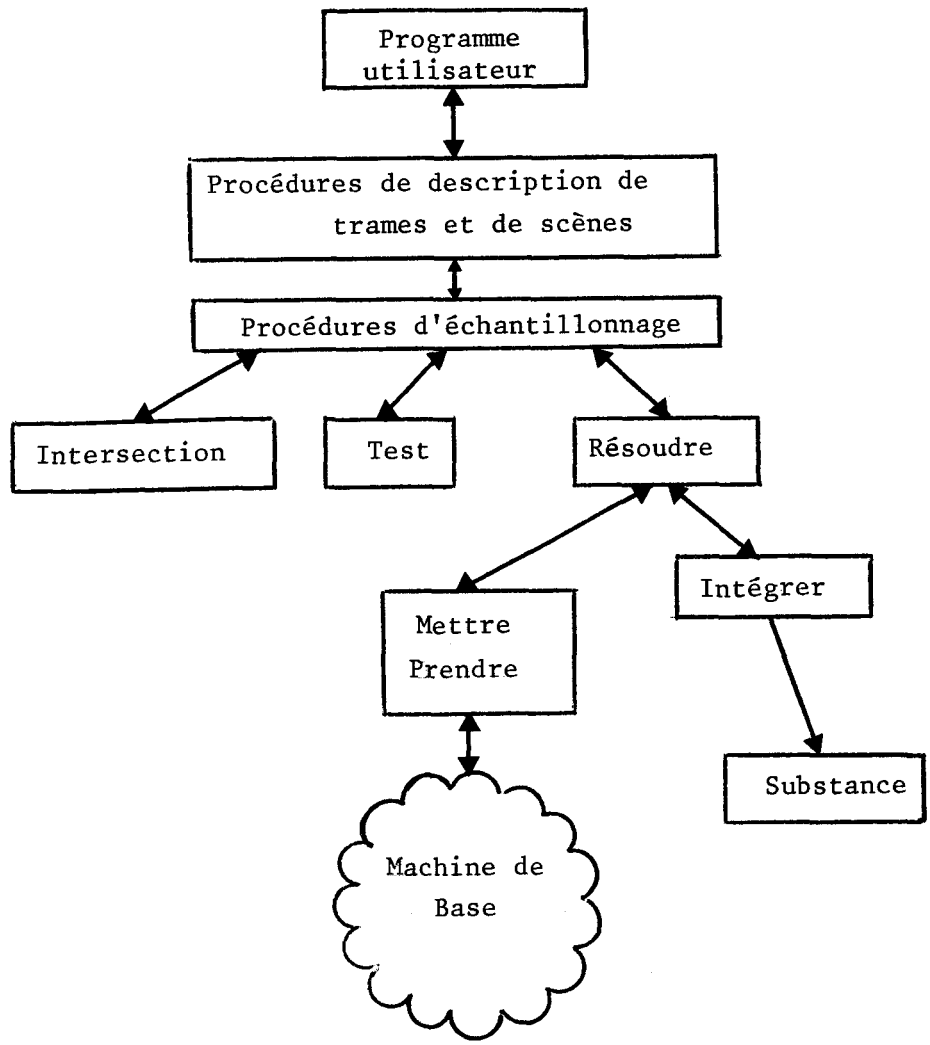


Figure 7

Cette figure montre les diverses procédures ainsi que leurs positions relatives. Chacune d'entre elles, ou plutôt chacun des blocs peut être redéfini dans le cadre d'une application particulière. Parmi les trois procédures, trois d'entre elles jouent un rôle très dépendant du contexte d'utilisation du noyau. Ce sont Resoudre, Integrer, et Substance(*1) Nous allons sur quelques exemples préciser leurs rôles respectifs

---

*1 En réalité, substance est un type, mais pour simplifier la présentation, nous considérerons qu'il s'agit d'une procédure unique.

Resoudre, comme nous l'avons vu dans le paragraphe précédent, doit déterminer la valeur à attribuer à tous les pixels d'une zone donnée. Cette procédure se charge donc de déterminer la contribution de chaque primitive et de combiner tous les résultats obtenus. La manière de procéder à cette combinaison "personnalise" un synthétiseur et ceci peut être rendu variable d'une application à l'autre. On peut noter à titre d'exemple, que c'est cette procédure qui utilise certaines valeurs d'attributs telles que les profondeurs dans les images de scènes tridimensionnelles, et qui peut ensuite utiliser le principe de la mémoire de profondeur pour faire des déterminations de visibilité.

Pour connaitre la contribution de chaque primitive, il est fait appel à la procédure Intégrer. Cette procédure est celle qui se charge de l'échantillonnage de la fonction caractéristique d'une primitive. C'est donc elle qui se chargera de la plupart des traitements dits d'"anti-aliasing" éventuels. Dans ses traitements, elle fait appel à Substance.

Substance est une procédure fournie par l'utilisateur dans la définition de chaque primitive, qui admet un objet de type point comme paramètre d'entrée, et retourne un autre objet de type point. Quelques exemples en sont donnés dans ce qui suit:

a). Une substance qui à tout couple (x,y) (point sans attribut) associe un triplet (x,y,c) où c est une constante, est une substance de facette uniforme.

b). Une substance qui à tout point (x,y,s,t) associe un point (x,y,R(s,t),V(s,t),B(s,t)) permet, entre autres, de réaliser des textures de la manière décrite par Catmull (Catm74) ou par Blinn et Newell (BlNe76). On peut en effet donner la couleur en un point d'une surface comme étant une fonction dépendant de deux paramètres variant sur cette surface.

c). Pour terminer, on peut prendre l'exemple d'une substance qui associe (x,y,R(s,t),V(s,t),B(s,t),Nx(s,t),Ny(s,t),Nz(s,t)) à tout point (x,y,s,t). Ce genre de substance permet de mettre en oeuvre une méthode de simulation de surface rugueuse, comme l'a décrit Blinn (Bl78.1, Bl78.2)

Ces trois exemples permettent de mettre mieux en évidence le rôle des attributs. On remarque que l'utilisateur fournit d'une part des points définis dans un espace d'attributs particulier, et d'autre part, une fonction transformant ces attributs en d'autres. Ce sont ces derniers attributs qui sont éventuellement rangés en mémoire d'images.

On remarque que la manière dont Résoudre, Integrer et Substance s'organisent permet de séparer diverses fonctions du synthétiseur, dans des modules indépendants, ce qui permet entre autres d'assurer une extensibilité dans de bonnes conditions.

3.6.Remarques complémentaires

L'organisation que nous venons de détailler répartit l'échantillonnage et l'"anti-aliasing" sur trois composantes distinctes. Resoudre peut faire des calculs précis sur la contribution de chaque primitive sur un pixel et Integrer peut se charger de procéder à des filtrages pour éviter certains problèmes liés à l'échantillonnage de Substance. On peut également envisager un filtrage dans Substance, mais ceci risque d'alourdir cette procédure qui doit pouvoir être manipulée par un utilisateur du noyau. De plus il faudrait fournir un paramètre supplémentaire concernant la distance d'échantillonnage. Ceci se rapproche de ce que nous allons détailler dans la suite de ce paragraphe.

On constate que Substance permet à un synthétiseur d'interroger les informations de l'utilisateur, et par conséquent il est possible de la définir d'une manière qui utilise certains aspects de la description hiérarchique des objets. Prenons par exemple la définition d'un mur d'immeuble avec des fenêtres et des portes. On peut imaginer que ce qui est fourni comme primitive de la scène est la géométrie du mur, accompagnée d'une substance décrivant les fenêtres et portes sur ce mur. Dans ce cas, si l'on peut communiquer à Substance une information sur la taille de la zone de travail de Integrer, il lui sera possible d'utiliser la structure hiérarchisée des informations.

4.Commandes pour le convertisseur

4.1.Introduction

Un convertisseur compose des images à partir d'images digitales et d'opérations diverses sur des images. Les commandes qu'il reçoit sont donc des commandes de visualisation et de combinaisons d'images. Nous avons vu dans le chapitre précédent qu'il existait de nombreuses fonctions possibles, et qu'elles étaient actuellement liées à la technologie des systèmes de visualisation. Nous supposerons dans les définitions qui vont suivre que la technologie n'impose aucune limitation, et nous examinerons les problèmes de mise en oeuvre plus loin dans ce chapitre. Le jeu de commandes proposé est volontairement limité et devrait pouvoir être étendu par la suite.

Une image sera une entité "logique" définie à partir de tranches de mémoire d'image et d'opérateurs, ou éventuellement à partir d'autres images. Lorsque après avoir effectué les opérations nécessaires sur un point, le convertisseur doit produire un signal pour le support de l'image, il y a projection de l'ensemble (0,...,n) des valeurs entières possibles pour la source sur l'intervalle (0,...,P) des valeurs possibles pour le signal de sortie.

On remarque que la définition qui précède implique que la valeur d'un pixel est une valeur entière, et non plus un vecteur. Ceci montre qu'une image sera maintenant considérée comme étant monochrome, les images couleur étant obtenues par des techniques que nous examinerons par la suite.

4.2.Définition et combinaison d'images

Dans les définitions qui suivent, nous allons utiliser un nouveau type qui ne sera pas défini car il dépend trop de la mise en oeuvre qui est faite, c'est le type image On peut considérer, dans le cadre de cette présentation qu'un objet de type image est un entier correspondant à un numéro d'image logique.

Toutes les commandes qui sont décrites dans ce paragraphe définissent des images logiques en fonction d'autres, ou par rapport à des tranches de mémoire d'image. Il n'existe à priori aucune limitation sur les combinaisons de telles commandes.

def-image (Imagel: image; T: Tranche)

Par cette commande, on indique que les valeurs des pixels de l'image logique Imagel seront obtenues par lecture du contenu de la tranche T. Les tranches manipulées par le convertisseur sont les mêmes que celles que connait le synthétiseur.

copier (Imagel: image; T: Tranche)

L'image Imagel est copiée dans la tranche T. Cette commande est en général utilisée pour stocker une image logique obtenue à partir de traitements divers sur d'autres.

Incruster (Imagel, Image2, Image3: Image; V: entier)

Cette commande définit les pixels de Image3 en fonction de ceux de Imagel et Image2 de la manière suivante:

```
si Image2(i,j) = V alors Image3(i,j) := Imagel(i,j)
                   sinon Image3(i,j) := Image2(i,j)
```

En pratique, cette commande permet de définir Image3 comme étant Image2 "sur fond" Imagel.

combiner(Imagel, Image2, Image3 : Image; a,b,c: entier; op: opérateur)

Par cette commande on définit les pixels de Image3 de la manière suivante:

```
Image3(i,j) := (a*Imagel(i,j) op b*Image2(i,j)) / c
```

op est un opérateur arithmétique ou logique pris dans un ensemble prédéfini (+, -, *, /, et, ou,..) Le premier rôle que l'on peut attribuer à cette commande est celui de permettre le "fondu" de deux images. Mais ce n'est pas sa seule utilisation, comme nous le verrons dans le chapitre suivant.

Dans le cas général, une définition d'image porte sur toute les images considérées, un point (i,j) d'une image source étant associé au point(i,j) de l'image objet. Il existe deux commandes permettant de changer cette association.

Deplacer (Image1, Image2: Image; dx,dy : entier)

Par cette commande, on définit Image2 de la manière suivante:

```
Image2(i,j) := Image1(i-dx,j-dy)
```

Les pixels de Image2 correspondant à des indices hors limite dans Image1 prennent une valeur nulle.

Grossir (Image1, Image2: Image; Fx, Fy: entier)

Par cette commande, on définit les pixels de Image2 comme suit:

```
Image2(i,j) := Image1(i/Fx,j/Fy);
```

Retrecir (Image1, Image2: Image; Fx,Fy :entier)

Cette commande est complémentaire de la précédente, et définit les points de Image2 de la manière suivante:

```
k := 0;
 pour x := 1 a Fx faire
      pour y := 1 a Fy faire
          k:= k + Image1(i*Fx+x,j*Fy+y);
Image2(i,j) := k/(Fx*Fy)
```

4.3.Tables de couleurs

Le rôle important de ces tables explique qu'il soit consacré un paragraphe complet aux commandes permettant de les utiliser. Avant de décrire la manipulation de ces tables, il est préférable de résumer sur un schéma (Figure 8) la manière dont est supposé être construit un convertisseur.

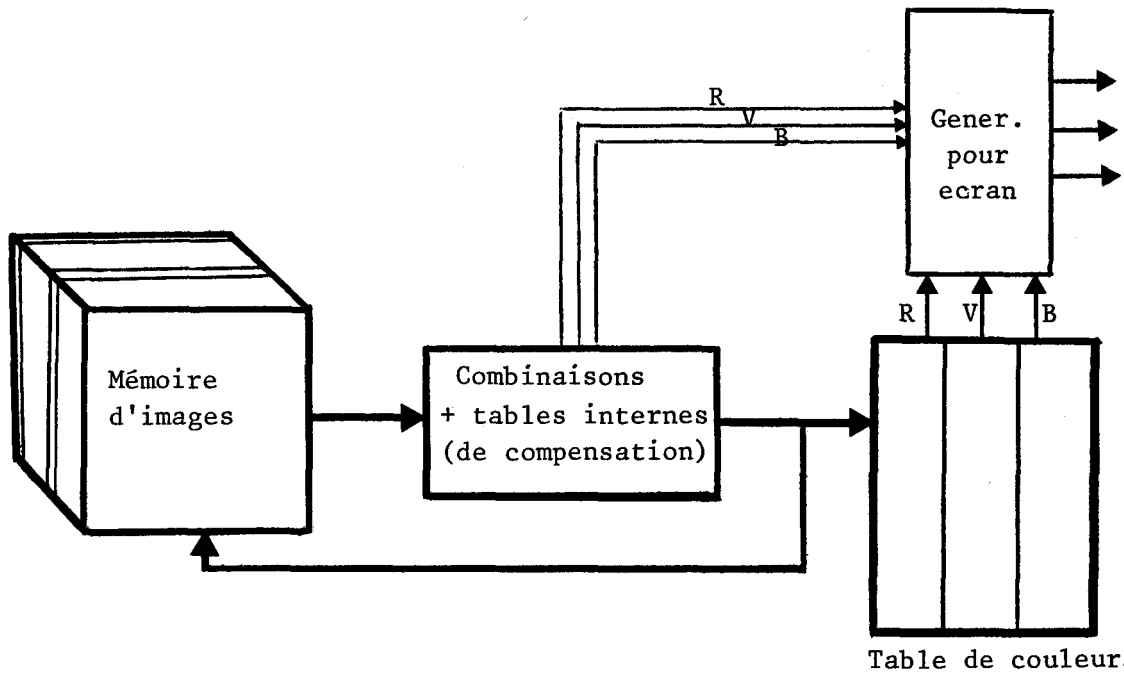


figure 8.

Ce schéma montre les places respectives des deux catégories de tables. d'une part, on trouve les tables de compensation, qui permettent de transformer une image logique en une autre, et d'autre part on trouve la table de couleurs qui n'est utilisée que lors de la visualisation. cette table de couleurs fournit trois valeurs <R,V,B> pour une seule valeur d'index. Ces deux types de tables se manipulent différemment.

Les tables de compensation sont utilisées par l'intermédiaire de la commande:

Compenser(Image1, Image2: image; T: tableau(1..N) de entier) Image2 est alors définie de la manière suivante:

```
Image2(i,j) := T(Image1(i,j));
```

La manipulation de tables de couleur est différente. Nous ne nous interesserons ici qu'à la manière dont on spécifie les valeurs de la table, son utilisation sera abordée au moment de l'étude des commandes de visualisation.

Les couleurs sont définies par des nombres entiers correspondant à des valeurs de Rouge, Vert et Bleu. Ces valeurs sont exprimées en centièmes d'intensité maximale (*1). Le problème du choix d'un autre système de

---

*1 Il s'agit d'une convention qui pourrait aisément remplacée par une

description de couleurs ne se pose pas au niveau d'un tel noyau, et relève plutot de la "modélisation". Il existe en effet des méthodes pour passer d'un système à un autre (JoGr78, Sm78.1), et elles peuvent être utilisées juste avant appel du noyau (voir Annexe 3 sur ce sujet).

La première commande que l'on trouve est:

Charger-table (n:entier; T: tableau(1..3,1..n) de entier);

Elle permet d'initialiser les n premiers éléments de la table à l'aide des valeurs contenues dans T.

On trouve ensuite:

Def-Couleur(n, r, v, b: entier);

Cette commande change le contenu de l'élément n de la table, en y rangeant les valeurs r, v et b.

4.4.Visualisation

Dans ce paragraphe nous allons examiner toutes les commandes qui permettent de faire apparaitre quelque chose sur l'écran. Nous allons donc détailler comment on utilise les images logiques et éventuellement la table des couleurs. Tout d'abord, nous allons voir comment se définit une image couleur.

Image-Couleur(Image1, Image2, Image3, Image4: Image);

Par cette commande, on indique que Image4 est une image couleur, ce qui signifie que ses pixels sont définis par des vecteurs à trois composantes, la première correspondant au rouge est prise dans Image1, la seconde dans Image2 pour le vert, et la troisième dans Image3 pour le bleu.

Pour obtenir une image sur l'écran, il faut utiliser la commande:

visualiser (Image: image);

Sur cette commande, Image est calculée en tout point et visualisée. Si l'image est une image couleur, il y a en réalité calcul de trois images, sinon une seule est calculée. C'est seulement à cette commande que les définitions et combinaisons d'images définies au préalable sont exécutées

---

autre telle que 1/64, 1/128 ou 1/256 par exemple.

effectivement, et ceci dans l'ordre où elles ont été données. Ceci a plusieurs conséquences qui seront mises en évidence sur les exemples suivants:

Prenons la séquence suivante:

```
def-image( Image1, T1);
def-image( Image2, T2);
incruster( Image1, Image2, Image3, v);
combiner( Image1, Image2, Image3, a, b, c);
visualiser( Image3);
```

Ce qui apparaitra sur l'écran sera la combinaison du contenu de T1 et de celui de T2, combiner ayant été défini après incruster, et ayant donc pris sa place dans la définition de Image3.

Prenons maintenant:

```
def-image( Image1, T1);
def-image( Image2, T2);
Incruster( Image1, Image2, Image3, v);
visualiser( Image3);
def-image( Image1, T3);
```

Après cette dernière commande, il y aura toujours sur l'écran le contenu de T2 dans T1, aucune modification n'étant faite jusqu'à l'ordre visualiser suivant.

Il existe également une commande permettant de demander la visualisation d'une image (nécessairement définie par une seule valeur par pixel) à travers la table de couleurs. Il s'agit de:

fausse-couleur(Image1: image);

Cet ordre indique qu'il faut visualiser Image1 en produisant les composantes R, V et B à l'aide de la table. Il a les mêmes effets que visualiser. En particulier, aucune modification de la table ne sera visible tant que n'aura pas été envoyé un autre fausse-couleur(*1).

---

*1 il est cependant fort probable que par la suite, il faudra rajouter une commande pour indiquer qu'il faut que la modification de table

Pour terminer, il y a la commande:

effacer;
qui permet d'enlever tout ce qui est sur l'écran.

5.Commandes diverses

Il existe de nombreuse autres commandes dont il serait souhaitable de disposer dans un noyau de système. Un premier ensemble important est celui des commandes permettant d'interroger par programme le système pour connaitre la configuration sur laquelle on se trouve. Il s'agit des commandes appelées habituellement INQUIRY dans les noyaux de systèmes graphiques. On constate lors de l'étude de ces noyaux, que le nombre de commandes de ce type devient très vite important, et l'étude détaillée des commandes d'interrogation ne présente que peu d'intérêt dans le cadre où nous nous situons.

D'autres commandes, souvent ignorées, concernent la synchronisation, ou plus précisemment l'attente d'évènements. On constate très souvent que la synthèse d'images est réalisée par un processeur indépendant de celui sur lequel s'exécute le programme d'application. C'est le cas lorsque l'on utilise un écran associé à un microprocesseur réalisant les fonctions élémentaires de synthèse par exemple. Dans ce cas, le programme utilisant le noyau ne sait pas toujours quand la synthèse s'est terminée, et ceci peut parfois présenter certains inconvénients. Il faut donc permettre au programme utilisateur de se mettre en attente de fin de synthèse. Ceci est possible à l'aide de la commande:

Attendre(s: signal)
qui met en attente le processus appelant jusqu'à ce qu'un évènement, spécifié par le paramètre s se produise.

Le type signal se définit comme suit:

---

apparaisse sans demande explicite, ne serait-ce que pour éviter de refaire toutes les opérations géométriques et de combinaison à chaque fois

```
type genre = (Synthétiser, Associer, Visualiser, Fausse-couleur);

type signal = struct
        G : genre;
        N : entier;
            fin
```

Lorsqu'une commande Fin-scène, Associer, Visualiser ou Fausse-couleur est appelée, il y a initialisation d'un évènement, défini par son genre, qui précise quel type de commande a causé l'initialisation, et par un entier, qui est soit le paramètre entier de Fin-scène ou Associer, soit fourni par le paramètre image de Visualiser ou Fausse-couleur. Le noyau de système active l'évènement et libère les éventuels processus en attente lorsque l'opération commandée est terminée. Le paramètre de type signal qui est fourni à la commande Attendre précise l'évènement que le processus attend.

6.Elements pour une mise en oeuvre

La mise en oeuvre d'un noyau de système peut se faire de nombreuses manières, suivant la configuration matérielle sur laquelle il doit s'éxecuter. Nous allons examiner les éléments à prendre en compte dans la suite de ce paragraphe, après avoir étudié une première expérience de mise en oeuvre.

6.1.Exemple de système élémentaire

Le premier chapitre a montré qu'il existe de nombreux types d'écrans, et nous allons étudier une mise en oeuvre réalisée avec une catégorie particulière d'entre eux. Il s'agit d'écrans qui n'ont pas besoin de raffraichissement, et donc principalement d'écrans bi-niveaux (*Vers, *TK14).

L'utilisation de ce type d'écrans élimine les contraintes de vitesse imposées au convertisseur, et également aux mécanismes d'accès à la mémoire d'image. Ceci permet donc au convertisseur d'être un programme (en FORTRAN dans le cas présent) et à la mémoire d'images d'être physiquement sur une mémoire secondaire (disque par exemple). Le synthétiseur est également un programme (FORTRAN également).

La mémoire d'image est organisée en tranches de 16 plans,(*1) la tranche n étant automatiquement destinée à recevoir l'attribut n d'un pixel. Le nombre de tranches, ainsi que le nombre de lignes et de colonnes des plans sont constants pour une même scène, mais peuvent être redéfinis par un ordre équivalent à Debut-Scène. L'utilisation de l'ordre Fin-scène permet de terminer proprement la synthèse d'une image, en forçant les mises à jour nécessaires sur le disque supportant la mémoire d'image.

Le synthétiseur comporte principalement un interpolateur linéaire, qui remplit des facettes à partir d'une description de contour polygonal, et réalise un échantillonnage primitive par primitive. Chaque sommet du polygone de définition est caractérisé par un vecteur (x,y,A1,...,An). L'accès à la mémoire d'images se fait par l'intermédiaire d'une procédure, qui joue le rôle de Resoudre dans ce que nous avons vu précedemment, et qui se charge de Prendre ou Mettre des pixels. Il existe de nombreuses procédures pouvant jouer ce rôle, et suivant le type d'application que l'on désire traiter, on peut en changer. Parmi les différentes possibilités, on trouve entre autres l'utilisation d'une mémoire de profondeur. Le premier attribut en un point est considéré comme étant la profondeur du point, et Resoudre fonctionne suivant le modèle schématisé dans ce qui suit:

```
Substance(point, point1);  (Calcul éventuel d'attributs
                                sans intégration)
point2.X := point1.X;
point2.Y := point1.Y;
Prendre(point2, trame);
 si point2.A(1) ≥ point1.A(1) alors Mettre(point1, trame);
```

On remarque que plus le premier attribut a une valeur élevée, plus on considère que le point est loin de l'observateur. On peut également mélanger point1 et point2 pour simuler des transparences, et ceci a également été réalisé dans la cadre de ce synthétiseur FORTRAN.

Ce qui précède montre qu'à aucun moment le synthétiseur ne connait la réalité physique de la mémoire d'images, si ce n'est qu'elle doit être accessible en lecture. En particulier, on peut imaginer que celle-ci ne se trouve pas sur un support physique uniforme (tout disque par exemple), mais

---

*1 Ce choix a été fait uniquement pour faciliter les accès à la mémoire secondaire, et certains plans sont souvent inutilisés.

sur plusieurs (disque pour la profondeur, mémoire de raffraichissement pour l'intensité,...). Les procédures Rendre et Mettre contribuent donc à rendre en partie le synthétiseur indépendant des matériels.

Le convertisseur, qui est indépendant du synthétiseur, joue un rôle limité dans cette mise en oeuvre. Il se contente de lire une des tranches de mémoire d'image, et de produire son image sur écran, en utilisant éventuellement une méthode de simulation de grisés(*1) si cela est nécessaire.

Un programme positionnant des facettes dans l'espace, et simulant un éclairement diffus a été réalisé, et la figure 9 montre des exemples d'images obtenues avec ce programme. La figure 10 a été obtenue en fournissant une image digitalisée au convertisseur, qui l'a visualisée sur imprimante electrostatique.

---

*1 Voir (JaJn76) par exemple

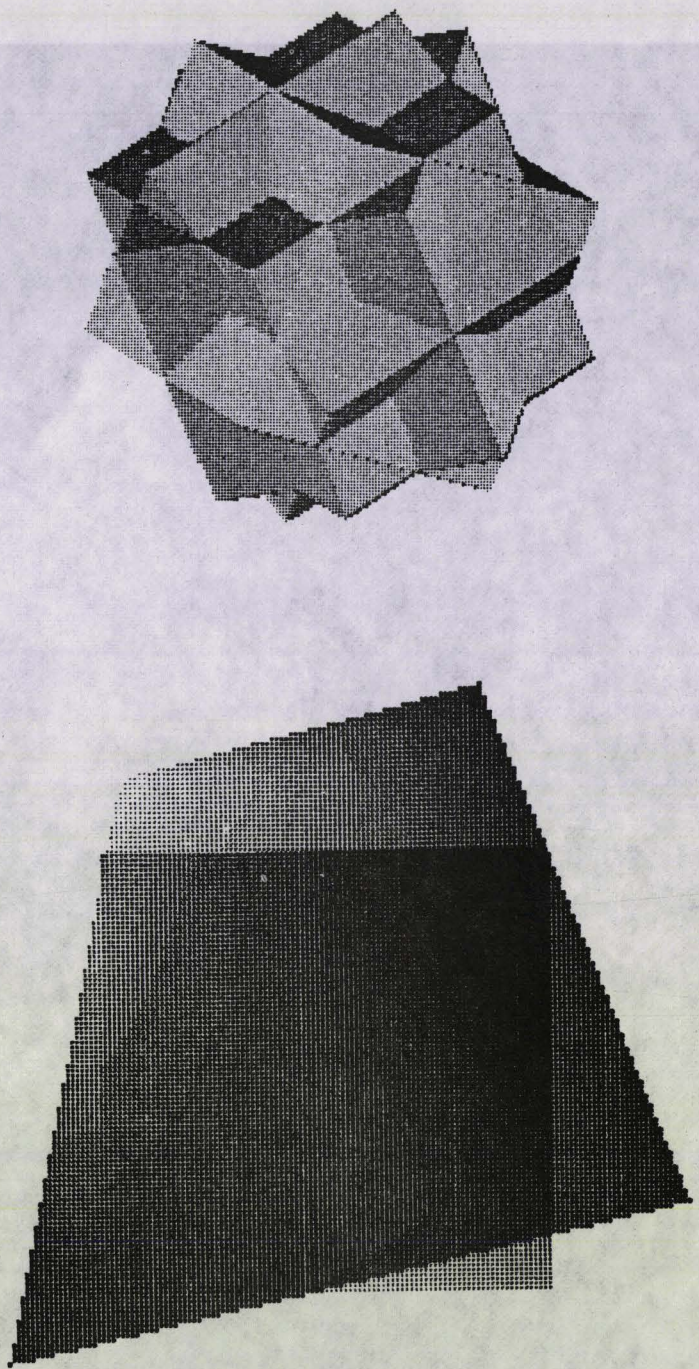


Figure 9.

Figure 10.

6.2.Mise en oeuvre d'un synthétiseur

Le Synthétiseur d'un système est l'élément qui est le moins soumis à des contraintes matérielles pour être mis en oeuvre. Nous avons vu que son seul lien avec le matériel se faisait par l'intermédiaire des procédures Mettre et Prendre. Comme nous l'avons déjà remarqué plus tôt dans ce chapitre, ces deux procédures peuvent simuler des fonctions que la mémoire d'images ne possède pas, en utilisant des mémoires secondaires, et également rendre homogène pour le synthétiseur, une mémoire d'images répartie sur divers supports physiques.

Cette approche n'est cependant pas toujours acceptable, principalement lorsque le système doit produire des images dans des conditions qui sortent de la "normale". C'est le cas lorsque l'on désire synthétiser une image en 1/30 de seconde, ou lorsque la trame a des dimensions trop importantes comme c'est le cas sur les imprimantes electrostatiques de grande dimension.

Dans le premier cas, il est souhaitable de grouper des ensembles d'accès à la mémoire d'images sous une seule commande, afin de diminuer les durées des opérations d'entrée-sortie, et il convient également d'organiser ces accès de manière à ce que les références ne se fassent pas de manière dispersée dans cette mémoire.

Dans le second cas, le grand nombre de pixels rend pratiquement impossible l'utilisation d'une mémoire pouvant contenir toute l'image digitale, et le plus souvent, la synthèse se fait bande par bande, en tenant compte du sens de défilement du papier.

Ce sont les principales raisons pour lesquelles les synthétiseurs sont de moins en moins bâtis sur le modèle de l'exemple du paragraphe 3.4. La plupart du temps, ils sont batis sur un modèle travaillant ligne à ligne (*1) ce qui permet alors de ne faire des accès à la mémoire d'images que ligne à ligne.

---

*1 Scan-line methods

6.3.Mise en oeuvre d'un convertisseur

Contrairement au synthétiseur, la mise en oeuvre d'un convertisseur est très dépendante du contexte matériel. On s'en rend compte aisément lorsque l'on compare un équipement utilisant les techniques vidéo à un équipement utilisant une imprimante. Les différences tiennent principalement au fait que dans le premier cas, la conversion est pratiquement toujours assurée par des processeurs cablés spécialisés, alors que dans le second elle l'est par des programmes(*1). On peut remarquer cependant, que sauf dans le cas où l'on désire travailler en "temps réel", il est possible de simuler les fonctions de conversion par programme, la mémoire d'image étant située sur des supports physiques accessibles par ce programme, et l'opération de visualisation consistant alors uniquement à transférer le contenu de cette mémoire vers l'écran ou sa mémoire de raffraîchissement. Ceci empêchera la plupart du temps à l'utilisateur de pouvoir se servir au mieux des caractéristiques spécifiques de son matériel.

On peut cependant esperer que dans un avenir plus ou moins proche, il deviendra suffisamment aisé de se définir et de se construire des modules matériels pour permettre de résoudre les problèmes liés au temps réel.

---

*1 Voir l'Annexe 4 pour le premier cas et le paragraphe 6.1 pour le second

IV. APPLICATION A LA VISUALISATION DE SCENES TRIDIMENSIONNELLES

1.Généralités

La visualisation de scènes tridimensionnelles avec simulation de phénomènes optiques réels fait intervenir un grand nombre de techniques de l'infographie, et de nombreux domaines d'application peuvent être considérés comme en étant des cas particuliers. C'est donc un domaine d'application qui présente un grand intérêt pour l'évaluation de ce qui a été proposé dans les chapitres précédents. Nous allons dans ce qui suit examiner comment un système de visualisation de scènes tridimensionnelles peut être bati autour de KISS.

La simulation du monde optique réel nécessite la définition de grandeurs très variées caractérisant la forme des objets, leur propriétés optiques, leurs positions, les conditions d'éclairement... On se rend compte aisément que toutes ces entités ne sont pas de même nature et qu'il convient d'effectuer une classification permettant de situer à quel endroit du système doit être pris en compte chaque type de caractéristique. Dans ce chapitre, nous examinerons tout d'abord le principal modèle utilisé pour la simulation des phénomènes optiques, et nous verrons qu'il est possible de séparer les caractéristiques des objets (reflectance.....) de celles caractérisant une scène (sources lumineuses.....) et de celles liées aux conditions d'observation (position de l'observateur.....). Nous etudierons ensuite comment la prise en compte de ce modèle peut être répartie entre un synthétiseur et un convertisseur, en utilisant diverses notions développées dans des études d'analyse de scènes. Nous montrerons pour terminer ce chapitre comment KISS peut être utilisé pour produire des images de scènes tridimensionnelles, en tenant compte des notions introduites.

2.Images de scènes tridimensionnelles

2.1.Généralités

Définir une scène tridimensionnelle consiste à définir un ensemble de points dans l'espace, à l'aide de modèles géométriques et à associer des caractéristiques optiques à ces points, à l'aide de modèles optiques. Pour visualiser une telle scène, il faut ensuite définir l'éclairement en chaque point et produire une image par projection des éléments de la scène sur un plan. L'opération que nous allons étudier en détail est cette projection, c'est à dire l'opération qui permet de calculer la fonction caractéristique d'une image à partir d'une scène tridimensionnelle. Lorsqu'il s'agit d'images en effet la projection n'est plus simplement géométrique comme pour les dessins. Il faut également projeter des caractéristiques des objets sur l'espace des attributs.

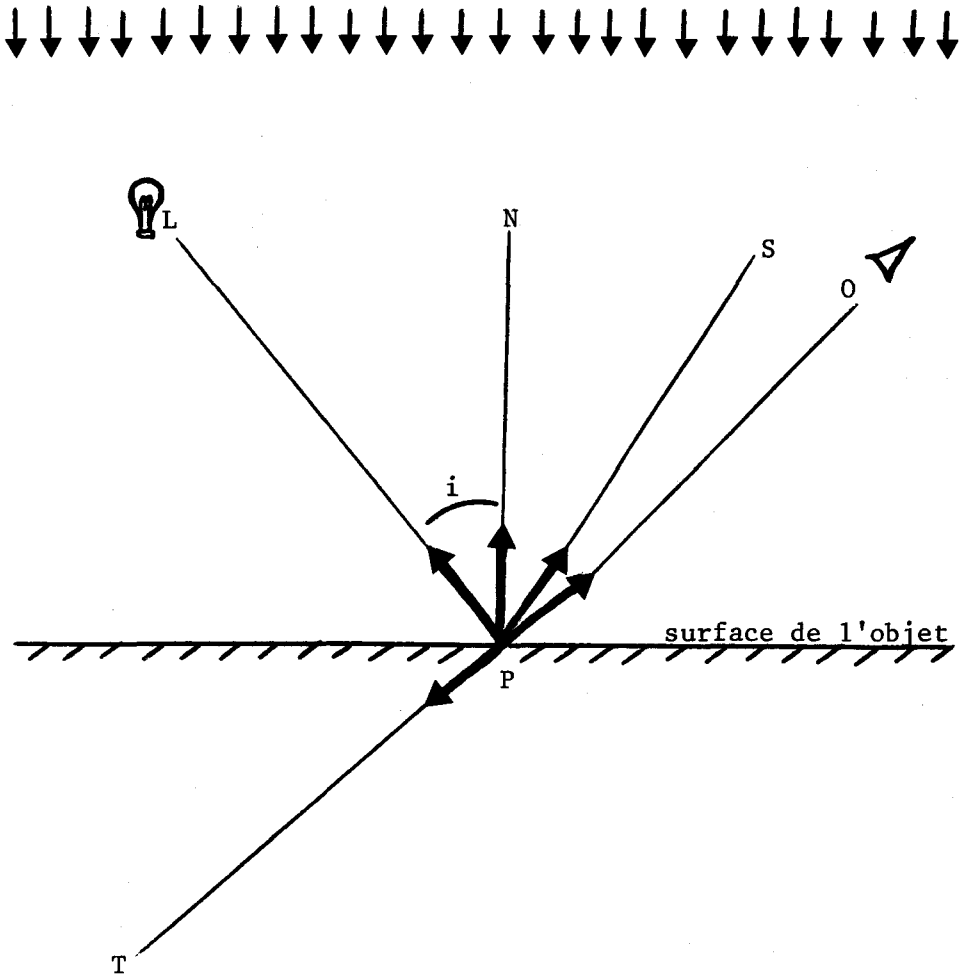
Dans ce paragraphe, nous allons tout d'abord détailler un modèle optique particulier, et examiner ensuite divers espaces d'attributs utilisables.

2.2.Modèle optique

Plusieurs modèles optiques ont été décrits pour produire des représentations simulant le monde réel avec le plus de "réalisme" possible. Celui qui a été proposé par Bui-Tuong-Phong (BuTP75) est le plus fréquemment utilisé, et nous nous bornerons à l'étude de ce modèle. Ce que nous verrons dans ce chapitre pourra être étendu aux principaux autres modèles, celui de Blinn (Blin77), et celui de Whitted (Whit79). Dans notre présentation, nous parlerons d'intensité en lumière monochrome, et il convient de transformer les équations scalaires en équations vectorielles lorsqu'il s'agit de lumière trichrome.

Les notations de la figure 11 seront utilisées tout au long de ce chapitre. Sur cette figure, les vecteurs en traits gras sont des vecteurs unitaires, et ne sont pas nécessairement tous coplanaires, contrairement à ce que montre la figure, qui ne se place dans ce cas particulier que pour des raisons de simplification du graphisme.

Lumière Diffuse



T

L = Source Lumineuse

N = Normale à la surface

S = Direction de plus forte reflexion

O = Observateur

T = Direction de transparence

figure 11.

Comme le montre cette figure, l'énergie lumineuse arrivant à l'observateur à partir d'un point donné, est due à trois causes. Tout d'abord, on considère que la scène est soumise à un éclairement diffus, c'est à dire que chacun de ses points recoit de l'energie lumineuse de manière uniforme suivant toutes les directions. A cela s'ajoute l'energie provenant de sources ponctuelles, en nombres quelconques. Enfin, si l'on suppose que l'objet que l'on regarde est transparent, il peut arriver une energie depuis l'"intérieur" de l'objet. L'équation (IV.1.) résume ce que nous venons de voir.

$$E = Ed + \sum Ep + Et \qquad (IV.1.)$$

Nous allons dans ce qui suit examiner chacune des composantes.

Prenons tout d'abord la composante due à la lumière diffuse. Elle est determinée par l'expression:

$$Ed = R.Id \qquad (IV.2.)$$

Id est l'intensité de la lumière ambiante.

On trouve dans cette équation l'apparition d'une des premières caractéristiques de la surface d'un objet, il s'agit du coefficient R (ou le vecteur R en imagerie couleur) qui est la reflectance au point considéré. C'est ce que l'on appelle traditionnellement la couleur de l'objet, mais ce terme est assez mal adapté au contexte dans lequel nous nous trouvons puisqu'il sert aussi bien à désigner ce que perçoit l'observateur que les propriété de la surface.

La composante liée aux sources ponctuelles est en réalité la combinaison de divers éléments. On trouve tout d'abord de l'énergie resultant de réflexion diffuse, c'est à dire d'une réflexion se faisant selon la loi de Lambert, qui précise que l'énergie incidente est réfléchie uniformément suivant toutes les directions, les différences d'intensité perçues venant des différences d'orientation de la surface refléchissante par rapport à la source emettrice. Cette variation en fonction de l'orientation se fait suivant une loi qui donne:

$$Er = R.Max(0.,\langle L.N\rangle).Ip \qquad (IV.3.)$$

R est la reflectance, <L.N> est le produit scalaire des vecteurs unitaires portés respectivement par la droite allant de la source au point considéré et la normale à la surface. <L.N> est donc le cosinus de l'angle de ces deux vecteurs. Si cette valeur est négative, la surface n'est pas éclairée par la source, et on considère alors que le cosinus est nul. Ip est l'intensité de la source ponctuelle. On trouve divers éléments intervenant dans le calcul, certains sont propres à l'objet dont on observe un point, d'autres sont liés à la scène, et plus précisemment aux "conditions de prise de vue". Nous reviendrons en détail sur leurs roles respectifs par la suite. En plus de cette reflexion diffuse, une source ponctuelle produit également une reflexion spéculaire, qui varie en fonction de la brillance d'une surface. Dans le modèle de Phong, cette composante est donnée par la relation:

$$Es = W(i).Max(0.,<S.O>^{**}n).Ip \qquad (IV.4.)$$

Le produit scalaire <S,O> mesure l'écart entre la direction de la droite allant du point observé à l'observateur et la direction de reflexion maximale, qui est la symétrique de la direction d'incidence par rapport à la normale. W(i) est un coefficient lié à l'angle d'incidence et qui dépend de la matière dont on suppose que l'objet est fait (Voir les exemples sur la figure 12. Enfin, l'exposant n du produit scalaire caractérise le "degré de brillance" de la surface, plus n est grand, plus la surface est brillante, c'est à dire que l'on s'approche d'un miroir.

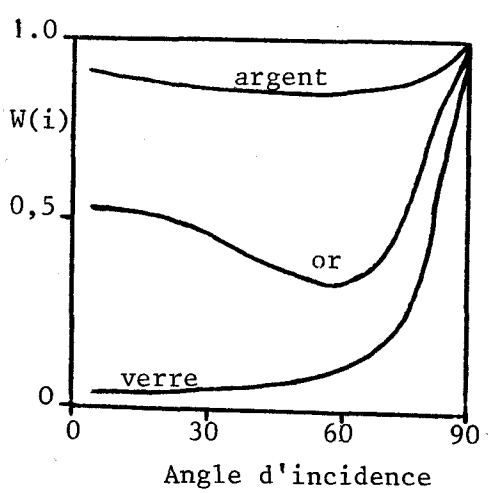


figure 12. d'après (NeSp79).

La dernière composante de l'energie transmise d'un point vers l'observateur est liée à la transparence de l'objet. Dans le modèle de Phong, il n'est pas tenu compte des phénomènes de réfraction (on suppose que le produit scalaire <T.O> est égal à -1). Ceci inportant dans la mesure ou une modification de transparence n'a pas d'influence sur la géométrie des rayons lumineux, mais uniquement sur leur intensité. Ceci n'est plus vrai dans le modèle proposé par Whitted, et ceci aura diverses repercussions que nous examinerons plus loin. Poue ce qui nous concerne, nous considérerons que la composante due à la transparence est donnée par la relation:

$$Et = T.It \qquad (IV.5.)$$

It est l'intensité arrivant sur le point, T est un coefficient de transparence.

On constate donc qu'il existe de nombreux éléments caractérisant les propriétés optiques d'une surface en un point, ainsi que les propriétés du monde dans lequel ce point est situé. Nous allons dans la suite de ce chapitre examiner comment il est possible de calculer des images digitales à partir de ces données.

2.3. Images intrinsèques

Le modèle que nous avons examiné est couramment utilisé pour visualiser des scènes tridimensionnelles. Le plus souvent, tout le travail est confié au synthétiseur qui se charge d"éclairer" les objets et d'appliquer les formules. Dans ce cas l'espace des attributs sur lequel est projeté la scène est un espace à une dimension (intensité) ou trois (Rouge, vert, bleu). Le rôle du convertisseur est alors très limité et ceci présente divers inconvénients, principalement en ce qui concerne l'extensibilité du synthétiseur, et la facilité de modification des conditions de visualisation.

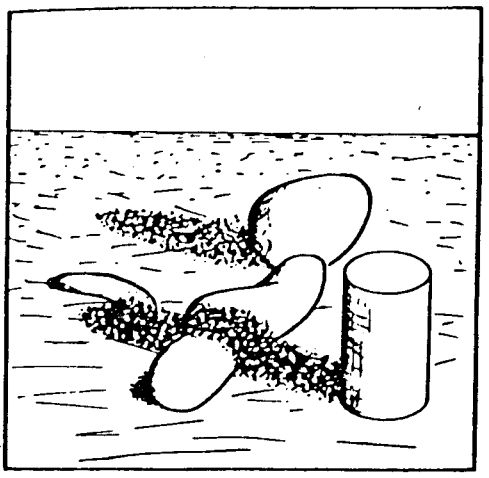
Nous allons voir dans ce paragraphe qu'il est possible de projeter la scène sur d'autres espaces d'attributs. Nous verrons qu'il est alors possible de mieux répartir les tâches entre synthétiseur et convertisseur.

L'utilisation d'attributs autres que des intensités lumineuses se fait déjà dans le domaine que l'on appelle habituellement analyse de scènes.

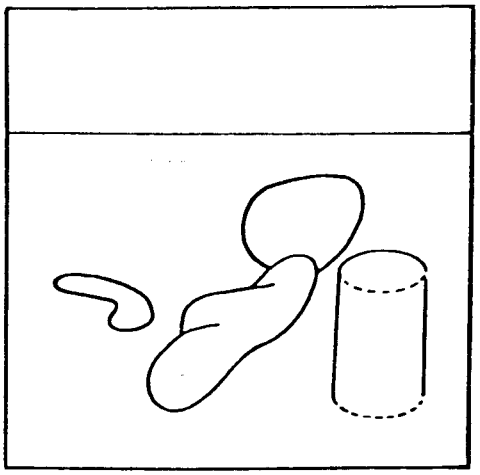
Dans ce type d'application, on cherche à résoudre un problème inverse de celui qui nous préoccupe. Il s'agit en effet de retrouver une description géométrique d'objets, en partant d'images, et en supposant qu'elles représentent des scènes dans lesquelles les lois optiques suivent un modèle fixé à priori. Les modèles utilisés sont très proches de celui que nous avons vu (Horn75, HoBa78). C'est dans ce cadre qu'a été introduite la notion d'image intrinsèque que nous allons utiliser (BaTe78, TeFT80).

Pour illustrer cette notion, prenons l'image d'une sphère homogène uniforme soumise à une illumination ponctuelle. Ce que nous verrons sera un dégradé d'intensité. Dans cette information, l'homogénéité de la sphère, qui est une caractéristique intrinsèque de cet objet sera masquée par les phénomènes optiques. Il en est de même pour un objet blanc soumis à un éclairement rouge, par exemple, la coloration n'est pas une caractéristique de l'objet, mais de l'éclairement. On constate donc que ce qui est perçu en un point d'une image digitale n'est pas simplement la valeur d'un attribut d'un objet ou de la scène, mais une combinaison de divers attributs. Pour l'analyse de scène, Barrow et Tenenbaum on développé des techniques permettant de calculer divers attributs à partir d'une intensité. En synthèse d'images, nous pouvons prendre la démarche exactement inverse, en faisant produire des images digitales dont les attributs ne sont pas nécessairement des intensités, et en faisant calculer les intensités par le convertisseur. Le synthétiseur calculera donc des images intrinsèques digitales.

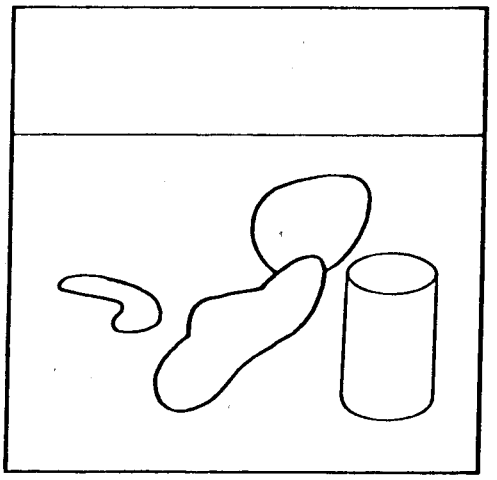
La figure 13 montre quelques images intrinsèques associées à une scène. Ces images ont été obtenues en projetant une propriété particulière de chacun des points visibles (distance par rapport à l'observateur, reflectance,...) et non une intensité lumineuse résultant de l'application d'un modèle optique. Sur cette figure, on a représenté par des traits les discontinuités et non les valeurs elles mêmes.



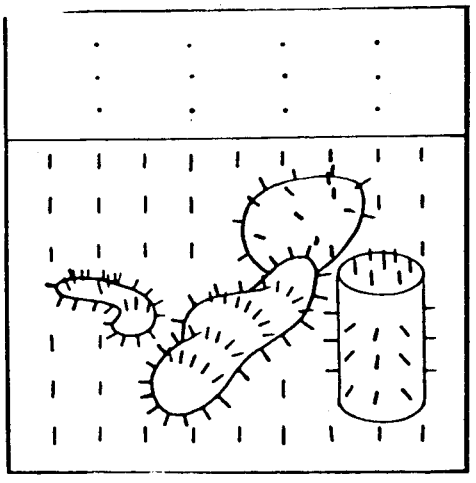
(a) Scène d'origine



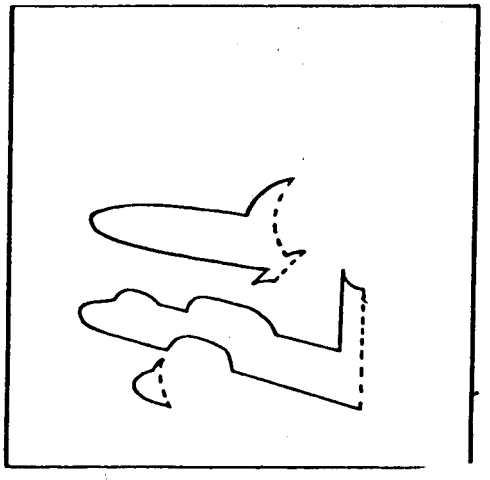
(b) Distance



(c) Reflectance



(d) Orientation (Vecteur)



(e) Eclairement

Représentations d'images intrinsèques associées à une scène

(D'après BaTe78)

figure 13.

2.4.Caractéristiques intrinsèques en un point

Avant d'étudier la projection des caractéristiques d'un point sur un espace d'attributs, nous allons tout d'abord examiner ces caractéristiques dans le cadre du modèle décrit précedemment.

On trouve tout d'abord la position du point, définie par les coordonnées (x,y) et également par une profondeur z. C'est à partir de cette position que l'on peut faire éventuellement les différents calculs de produits scalaires, en utilisant la position de l'observateur et celle des sources lumineuses ponctuelles.

Une autre caractéristique géométrique importante est la direction de la normale à la surface sur laquelle se trouve le point. Elle se définit par un triplet <Nx,Ny,Nz>, mais l'utilisation de directions normalisées permet de se contenter du couple <Nx,Ny>, Nz se calculant par la formule Nz = 1-SQRT(Nx*Nx + Ny*Ny).

Les autres caractéristiques ne sont plus géométriques, mais optiques. On trouve la reflectance qui est un nombre (en N/B) ou un triplet (en couleur), le degré de "spécularité", qui est l'exposant du produit scalaire <S.O> dans le modèle, et le coefficient de transparence qui est un nombre (en N/B) ou un triplet (en couleur).

Tous les autres éléments entrant dans le calcul de l'intensité en un point ne sont plus uniquement dépendants de l'objet visualisé, mais également des conditions d'éclairement de cet objet. C'est le cas du coefficient de spécularité W(i) et également des positions de l'observateur, des diverses sources lumineuses, ainsi que des intensités de lumière diffuse, ponctuelle ou de transparence.

On note enfin des éléments qui dépendent de la scène, c'est-à-dire des positions relatives des objets, ainsi que de l'éclairement, il s'agit des ombres portées et des reflexions secondaires

3.Utilisation du noyau

La répartition du travail entre synthétiseur et convertisseur dépend de l'espace d'attributs choisi pour les images digitales. Dans le cas habituel, tout le travail est assuré lors de la projection des caractéristiques intrinsèques des objets sur un espace unidimensionnel d intensités ou tridimensionnel de couleurs. Nous allons, dans les paragraphes qui suivent, examiner d'autres espaces sur lesquels peuvent se projeter les propriétés des objets, ainsi que la manière dont celà peut être mis en oeuvre en utilisant le noyau décrit dans le chapitre précédent.

3.1.Espace Reflectance + Normale

Prenons tout d'abord la cas d'une application qui ne nécessite de mettre en oeuvre que le calcul d'intensités dues à la lumière ambiente, et à la reflexion diffuse d'une source ponctuelle unique, supposée se trouver à l'infini, le tout étant dans un espace monochrome. La formule donnant l'intensité en un point (i,j) est:

$$E(i,j) = R(i,j)*(Id + Ip*Max(0.,<L.N(i,j)>)) \qquad (IV.6.)$$

Ip et Id sont constantes en tout point de l'espace de la scène, et L est également constant puisque la source lumineuse est supposée à l'infini. Les seuls éléments qui varient d'un point à l'autre sont donc R(i,j) la reflectance en (i,j), et (Nx(i,j),Ny(i,j),Nz(i,j)) qui est la direction de la normale en ce point(*1). Nous pouvons donc utiliser un vecteur attribut (R,Nx,Ny,Nz) pour caractériser un objet en un point.

Une idée consiste à faire manipuler par le synthétiseur des réseaux de points (X,Y,A1,A2,A3,A4), en associant une tranche à chaque attribut. Le convertisseur aura donc à sa charge le calcul de la formule (IV.6) en tout point. On constate aisément qu'il devient possible de modifier les conditions d'éclairement où même le modèle sans devoir refaire une synthèse.

La suite de ce paragraphe indique la manière dont le convertisseur peut procéder pour calculer l'intensité en un point.

---

*1 Nous n'utiliserons pas dans un premier temps le fait que N est un vecteur normalisé et que Nz peut donc se déduire de Nx et Ny

En mettant la formule IV.6. sous une autre forme, on fait apparaitre plus nettement les opérations du convertisseur.

```
E(i,j) = R(i,j)*(Id + Ip* ABS(Lx*Nx(i,j) + Ly*Ny(i,j) + Lz*Nz(i,j)))
```

De cette formule, on peut déduire la suite de commandes suivante:

```
var R,Nx,Ny,Nz : image;
A,Unité : image;
T1,T2,T3,T4 : tranche;
. . . . . .
(* T1, T2, T3 et T4 sont définies de manière conforme à ce que
 le synthétiseur est supposé avoir fait.*)
. . . . . .
def-image(R,T1);
def-image(Nx,T2);
def-image(Ny,T3);
def-image(Nz,T4);

combiner(Nx,Ny,A,Lx,Ly,1,somme);
combiner(Nz,A,A,Lz,1,1,somme);
.............détermination de valeur absolue(*1)
combiner(Unité,A,A,Id,Ip,1,somme);
combiner(R,A,A,1,1,1,multiplication);
visualiser(A);
```

Lorsque plusieurs sources sont à prendre en compte, on peut répéter ces opérations autant de fois qu'il le faut, et la réflexion spéculaire peut être calculée d'une manière identique, en utilisant par exemple une table pour obtenir le coefficient de spécularité associé à un angle donné, c'est à dire à une valeur de <L.N> donnée. Seule l'exponentiation pose un problème dans le noyau précedemment défini, mais il n'est pas insurmontable si l'on se contente de puissances multiples de deux.

Une autre manière de procéder consiste à utiliser une table de couleurs.

---

*1 cette determination peut se faire à l'aide d'une incrustation (définie d'une manière un peu différente de celle que nous avons vu), ou d'une autre fonction non précisée dans le chapitre précédent.

comme cela a été suggéré par Sloan et Brown (SlBr79). Le principe de cette méthode consiste à faire produire des attributs R, Nx et Ny par le synthétiseur et à faire calculer une valeur unique par le convertisseur à partir de ces trois valeurs. Cette valeur sert ensuite à indexer une table dans laquelle a été rangée l'intensité associée à chaque triplet (R,Nx,Ny). Le contenu de cette table est fourni au convertisseur par le programme d'application. On constate à nouveau qu'il est possible de synthétiser une nouvelle scène sans devoir faire des calculs de couleur, ou de modifier un modèle d'éclairement sans modification du synthétiseur, ni même d'une image déjà synthétisée. Cette approche par tables de couleurs présente l'avantage d'éliminer le problème des opérations complexes pour le convertisseur, mais présente par contre l'inconvénient de ne donner que peu de possibilités de couleurs et des dégradés de couleur parfois trop quantifiés. Ceci est du a la taille limitée que l'on donne aux tables.

3.2.Autres espaces

D'autres espaces peuvent être définis, en utilisant par exemple des attributs de distance (mémoire de profondeur), d'ombre ou de texture.

L'attribut distance, qui est parfois utilisé par le synthétiseur pour faire des déterminations de visibilité, peut servir à délaver les couleurs des points éloignés de manière à donner un rendu plus "réaliste" des paysages, tels que ceux que l'on rencontre dans des simulateurs par exemple. Cet attribut n'est cependant pas interprété par le convertisseur, dans le cas général.

On peut également utiliser un attribut Ombre (ou Eclairement si l'on reprend la notation de la figure 13.). Il est possible en effet de calculer, en tout ou rien, l'ensemble des ombres portées, dans une scène, pour une source lumineuse donnée. On peut donc envisager de mémoriser tous les points visibles mais non éclairés de manière à utiliser cette information lors d'un calcul d'intensité. De plus, on peut conserver une telle "image d'ombres" pour chaque source, ce qui permet éventuellement, si l'on modifie l'une des sources de ne pas devoir resynthétiser toute l'image.

Un attribut texture peut lui aussi être employé, et permettre, à l'aide d'une table par exemple, de répartir la fabrication de texture entre le synthétiseur et le convertisseur, en permettant de modifier certaines

caractéristiques d'une texture sans refabriquer toute l'image. Par exemple, on peut envisager qu'une surface visualisée soit un ensemble de carreaux de l'espace ayant chacun une texture définie par un ensemble de 256 valeurs (16*16). Dans ce cas, on peut affecter à chaque point de la surface une valeur repérant sa position dans un carreau, et cette valeur, par l'intermédiaire d'une table à 256 entrées permettra éventuellement de donner des textures à des surfaces.

Ces divers exemples montrent qu'il est possible de définir les traitements pour la calcul d'une image, d'une manière modulaire, permettant de les combiner de diverses manières, sans devoir refaire un synthétiseur ou un convertisseur à chaque fois. Par exemple, on peut envisager de remplacer le modèle d'éclairement de Phong par celui de Blinn sans devoir changer autre chose que la partie de programmation du convertisseur réalisant la formule mathématique du modèle. En particulier, un calcul éventuel d'ombres portées n'aura pas à être modifié, ni une méthode de fabrication de texture. De plus, le synthétiseur ne se rend compte de rien, et il ne se préoccupe que de calculs géométriques, d'interpolation, et d'échantillonnage.

3.3.Traitements et modèles géométriques

Nous avons vu dans les chapitres précédents, que le modèle géométrique retenu est limité. En effet, le synthétiseur ne connait que des facettes polygonales planes, dont l'intérieur est determiné par interpolation linéaire à partir des contours. Ce modèle peut parfois s'avérer insuffisant, et on peut alors vouloir en utiliser d'autres.

Les deux autres modèles le plus souvent rencontrés en synthèse d'images à l'heure actuelle sont les ellipsoïdes et les surfaces paramétriques bicubiques.

A de rares exceptions près (Herb80) les ellipsoïdes, qui sont le plus souvent des sphères, sont utilisées dans des applications liées à la représentation de grandes molécules, où la vitesse d'ombrage d'une sphère est très importante puisqu'une scène en comporte plusieurs milliers. Ce type de système est donc peu concerné par les structures générales et extensibles.

Les carreaux bicubiques par contre sont souvent employés, et ne peuvent

donc pas être négligés. On notera que l'utilisation de ces représentations peut dans un premier temps être reportée au niveau de la modélisation, puisque plusieurs articles ont montré que la décomposition de bicubiques en carreaux bilinéaires était simple et donnait des resultats très satisfaisants (LCWB80, Clar79). Cependant, la décomposition se faisant d'une manière liée à la dimension des carreaux par rapport au tramage, il convient de l'intégrer au synthétiseur si l'on veut utiliser les avantages liés aux descriptions hierarchiques d'objets.

V. CONCLUSIONS ET PROLONGEMENTS POSSIBLES

1.Remarques sur la parallélisation

La synthèse et la conversion sont deux traitements qui répètent les mêmes opérations sur un très grand nombre d'objets différents, que l'on peut facilement rendre indépendants les uns des autres. Il est donc assez naturel de penser à confier ces travaux à des processus parallèles si l'on désire améliorer les temps de réponse d'un système. Dans ce paragraphe, nous allons examiner divers aspects de la parallélisation des processeurs d'un système de synthèse d'images. Nous examinerons séparément le cas du synthétiseur et celui du convertisseur, qui présentent comme nous le verrons, de grandes différences.

Si l'on reprend l'exemple de procédure de synthèse du paragraphe 3.4. on constate qu'il est facile de répartir le travail entre plusieurs processus indépendants. On remarque que les quatre appels de la procédure Synthétiser se font indépendamment les uns des autres, et que leurs executions peuvent se faire en parallèle. Ceci suggère immédiatement l'idée de faire travailler des processus executant la même procédure, en parallèle sur des zones de mémoire d'images différentes. La figure 14 montre quelques exemples de répartition de la mémoire d'images entre différents processus.

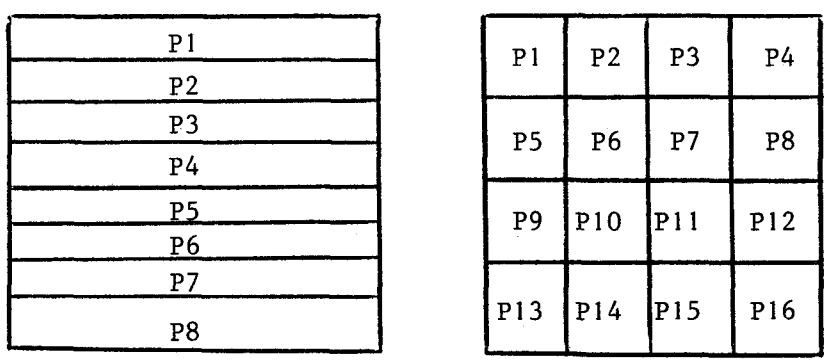


Figure 14.

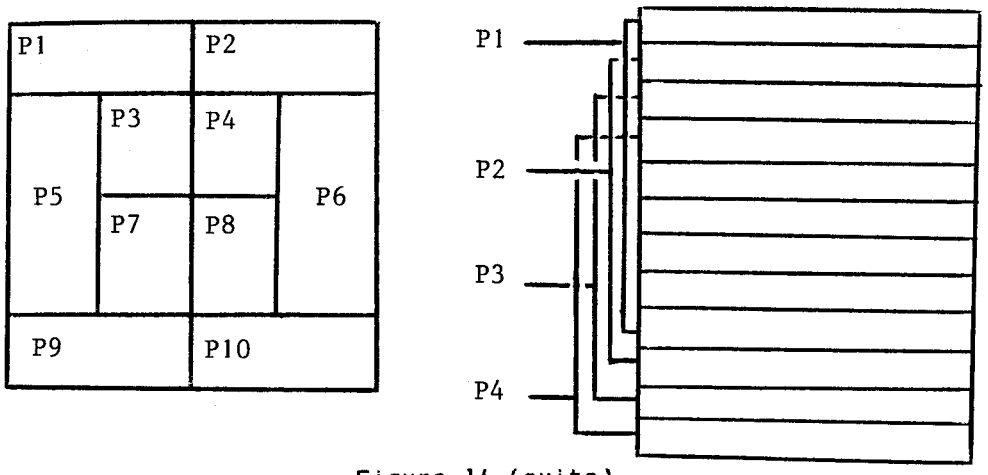


Figure 14.(suite)

La répartition de la mémoire d'images entre les différents processus se fait suivant des critères divers. On peut soit tenir compte de contraintes matérielles, soit d'estimations de charge des processus. En ce qui concerne ce dernier point, on peut noter quelques premiers travaux comme ceux de Kaplan et Greenberg (KaGr79) ou Fuchs et Johnson (Fuch77, Fujo79).

L'évolution technologique actuelle rend envisageable l'utilisation de matériels multi-processeurs, et on peut donc penser à faire executer un seul processus par processeur. L'architecture la plus souvent rencontrée est celle qui est schématisée sur la figure 15.

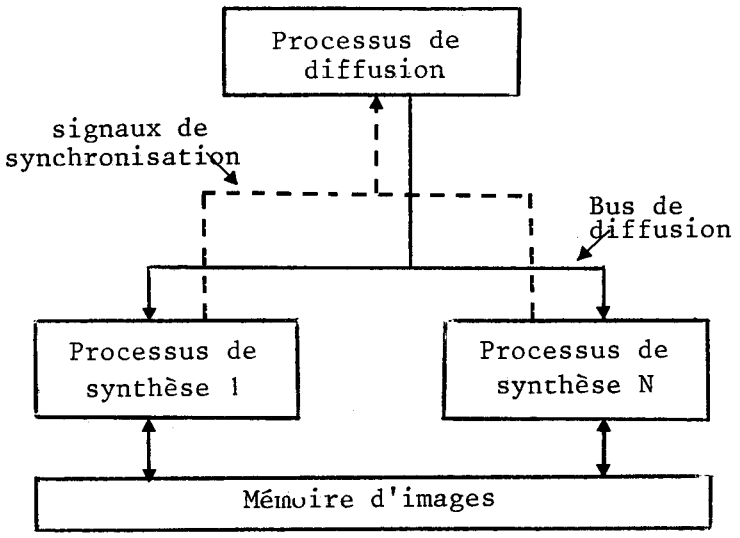


Figure 15.

Le Processus de diffusion se charge simplement d'emettre sur un bus l'ensemble des informations concernant la scène à visualiser, et tous les

autres processus se contentent de recevoir ces informations, chacun se chargeant localement de determiner ce qui le concerne. Les différents processus de synthèse accèdent également à la mémoire d'images. On trouve parfois également une ligne permattant aux processus de synthèse d'emettre des signaux en direction du processus de synthèse. Ceci est nécessaire lorsque l'on désire utiliser un signal de "fin de synthèse" comme nous l'avons vu dans le chapitre décrivant le noyau.

Comme chaque fois que plusieurs processeurs se partagent un même moyen de communication, il se pose des problèmes de partage. Ceux-ci sont cependant assez limités dans le contexte qui nous intéresse. On remarque tout d'abord que très souvent, il n'y aura qu'un seul émetteur sur le bus de diffusion, et que par conséquent, il n'y a pas de conflit d'accès à ce niveau, sauf si la synchronisation utilise le même dispositif de communication. En ce qui concerne le partage de la mémoire d'images, on peut remarquer que si les processus travaillent sur des parties distinctes de la mémoire, il ne se posera pas de problèmes entre eux. Cependant, si ils travaillent en parallèle avec un convertisseur raffraichissant un écran vidéo par exemple, il se posera alors des problèmes d'accès entre synthétiseurs et convertisseur, et ce problème peut ne pas être simple à résoudre.

En ce qui concerne la parallélisation dans un convertisseur, on ne peut pas aborder le problème de la même manière que précedemment. Le plus souvent, en effet le convertisseur doit traiter les pixels les uns après les autres, dans un ordre fixé en général par les contraintes de balayage, ou le sens de défilement d'un papier par exemple. La structure de processus parallèles indépendants n'est donc plus adaptée, mais on doit s'orienter vers une structure "pipe-line". C'est ce que font actuellement la plupart des constructeurs de matériels destinés au traitement d'images(*1), c'est à dire à des applications pour lesquelles ce qui est appelé conversion dans cette thèse est très important.

On pourrait se ramener à une situation comparable à celle d'un synthétiseur en séparant la mémoire d'images en deux parties distinctes, la première recevant les attributs depuis le synthétiseur, l'autre recevant des informations <R,V,B> pour pour l'écran, on peut alors faire effectuer une grande partie du travail de conversion par une batterie de processeurs

---

*1 Voir par exemple l'Annexe 4.

situés entre les deux étages de mémoire d'image, ces processeurs n'ayant plus à travailler dans un ordre imposé par le balayage.

2.Prolongements envisagés

Le noyau a été conçu de manière à permettre de mener divers développements et expérimentations dans de bonnes conditions. Cette facilité est principalement due à la structuration qui a été retenue pour ce noyau. Celui-ci permet de modifier certaines parties d'un système en limitant les repercussions sur les autres composantes. Il est donc possible d'envisager divers prolongements à cette étude, chacun portant sur une partie spécifique d'un système.

En ce qui concerne la hierarchisation des descriptions de scènes, on peut penser à décrire une procédure de synthèse qui tienne compte de cet aspect, par exemple en changeant, par rapport à l'exemple du paragraphe III.3.4, la manière de "prendre une primitive" et de calculer l'intersection de son domaine avec la zone de travail.

De manière indépendante, diverses études peuvent être menées sur la manière de produire des textures pour des objets tridimensionnels, aussi bien que pour des objets plans. Dans ce genre de problèmes, il se pose non seulement un problème de description et de représentation interne de modèles de textures, mais également un problème de qualité de résultat. Les textures sont en effet très sensibles aux dégradations dues à l'aliasing, qui apparait dès que l'on utilise des transformations géométriques pour positionner des objets, ou pour les visualiser (rotations, perspectives,...).

On peut remarquer que la description d'objets graphiques pour la synthèse d'images nécessite la définition de nombreuses entités qui ne sont pas uniquement des coordonnées de points. Dans les systèmes de production de dessins, la description d'informations non géométriques n'a été que peu abordée, il suffit pour s'en convaincre d'examiner les méthodes de définition de style de tracé (continu, tireté,.....). On constate que le plus souvent, l'utilisateur ne peut que donner une référence dans un catalogue prédéfini, et qu'il lui est rarement possible de se décrire un nouveau style de tracé. On trouve des contre exemples à cette affirmation, mais ils sont assez rares, et les outils de définition sont peu satisfaisants (voir Hage77, ou *Bens par exemple). On imagine donc

l'ampleur du problème lorsqu'il s'agit de décrire des images, et le besoin d'études approfondies sur les langages ou outils de description. On peut également étendre ce problème à la description de la dynamique de ces objets, mais celà est du ressort d'autres types d'études.

Une dernière série d'études est liée à la parallélisation des traitements. Nous avons vu que la répartition de mémoire d'images entre différents processeurs pouvait se faire de diverses manières, en tenant compte par exemple de la charge de travail de chacun. Il est donc nécessaire de procéder à des évaluations de cette charge en procédant à des mesures sur diverses catégories de scènes et d'images. De même, il peut être intéréssant de procéder à des mesures sur les diverses valeurs d'attributs utilisées dans une image de manière à développer des techniques permettant de n'utiliser que peu de bits pour définir un attribut, en utilisant une table de compensation et allocateur d'entrées dans cette table. Si l'on remarque que certaines images couleur de bonne qualité sont visualisables en n'utilisant que 256 valeurs et une table convenablement définis, il est probable que ce genre de techniques permettra de réduire énormément la taille des informations décrivant une image.

3.Conclusions.

Ce qui a été présenté dans cette thèse montre à quel point il est nécessaire de reprendre l'étude des organisations de systèmes graphiques lorsque l'on desire produire des images et non plus des dessins. On constate que les changements sont dûs principalement au fait que l'on ne se préoccupe plus exclusivement (ou presque) de géométrie, et que la "substance" des objets prend une place plus importante qu'auparavant. Par ailleurs, contrairement au systèmes de dessin, les entités manipulées lors de la synthèse d'une image ne sont plus de nature semblable, on trouve des objets géométriques, et des pixels, alors que dans le passé, on manipulait des traits d'un bout à l'autre. Ceci contribue également aux changements nécessaires.

Si on examine la répartition des fonctions entre synthétiseur et convertisseur, on constate que le premier se charge principalement des traitements géométriques, alors que le second peut assurer la plupart de ceux liés à cette substance. C'est une des raisons pour lesquelles ce second processeur, qui n'apparait pas dans la plupart des systèmes

graphiques de production de dessins, devient aussi important. On trouve toutefois dans certains logiciels de dessin récents, une première approche de cette séparation de taches entre divers types de processeurs, il s'agit principalement de GRIGRI (LeLM76, LeLM77) et de GKS (GKS79).

L'importance croissante du rôle du convertisseur peut également être reliée à l'évolution des technologies, qui permettent de plus en plus facilement de bâtir des matériels spécialisés. Dans les applications de traitement d'images, où le principal outil est un convertisseur, on voit maintenant apparaitre des machines dont le matériel est capable de réaliser des opérations de plus en complexes, et on peut envisager qu'il en soit de même pour les machines "graphiques". Il est probable que l'étude d'architectures utilisant une telle répartition des taches, et permettant de tenir compte des découpages en modules qui ont été présentés dans cette thèse puisse permettre de définir ces nouvelles machines.

ANNEXES

Les annexes qui suivent sont destinées à donner des compléments d'information sur divers sujets abordés dans cette thèse.

L'annexe 1 récapitule l'ensemble des commandes du noyau qui a été décrit dans le chapitre III.

L'annexe 2 donne un exemple classique d'algorithme d'échantillonnage de segments de droite et une version modifiée de cet algorithme permettant d'obtenir des vecteurs sur lesquels l'effet de "marches d'escalier" est atténué.

L'annexe 3 présente une partie des commandes disponibles pour décrire des images sur un terminal vidéo. Le jeu de commandes est représentatif de ce que fournissent la plupart des terminaux actuels destinés aux "graphiques", et des outils de description fournis par les extensions de systèmes graphiques traditionnels.

L'annexe 4 présente quelques aspects d'une console dite de "traitements d'images", et permet de fixer les idées sur les fonctions dont on peut disposer sur les convertisseurs actuels.

ANNEXE 1: Récapitulation des commandes de KISS

1.Commandes au Synthétiseur

1.1.Définitions de trame

def-fenetre(fxmin,fymin,fxmax,fymax: entier; T: trame)

def-cloture(cxmin,cymin,cxmax,cymax: entier; T: trame)

def-tranche(n,t1,t2,v1,v2: entier; T:trame)

ini-tranche(n,v: entier)

1.2.Primitives et attributs

fin-scène(n: entier)

debut-scène(T:trame)

associer(n:entier; D: Domaine; T:Transformation; R: ListePoints; S:Substance)

ini-réseau(nom,ndim: entier)

ajouter-point(nom: entier; élément: Point)

2.Commandes au convertisseur

2.1.Définition et combinaisons d'images

def-image(image1: image; T: tranche)

copier(image1; image; T: tranche)

incruster(image1,image2,image3: image; v:entier)

combiner(image1,image2,image3: image; a,b,c: entier; op: opérateur)

deplacer(image1,image2: image; dx,dy: entier)

grossir(image1,image2: image; Fx,Fy: entier)

retrecir(image1,image2: image; Fx,Fy: entier)

2.2.Tables de couleur

compenser(image1,image2: image; T: tableau(1..N) de entier)

charger-table(n: entier T: tableau(1..3,1..N) de entier);

def-couleur(n,r,v,b: entier)

2.3.Visualisation

image-couleur(image1,image2,image3,image4 :image)

visualiser(image: image)

fausse-couleur(image: image)

effacer

3.Synchronisation

Attendre(s:signal)

ANNEXE 2: Algorithmes de tracé de vecteurs

De nombreux algorithmes ont été développés pour échantillonner des traits, mais la plupart font un échantillonage primitive à primitive, en tout-ou-rien. Cela signifie que les traits sont supposés sans épaisseur, et que si un pixel est traversé par le trait il prend l'attribut affecté au trait, sinon il conserve sa valeur initiale. Ce mode de travail provoque l'apparition de "marches d'escalier", et n'est donc pas toujours satisfaisant. Dans la littérature, on trouve plusieurs images obtenues à l'aide d'algorithmes améliorés, mais ceux-ci sont rarement publiés. Cet annexe présente un algorithme modifié pour produire des traits digitalisés sur lesquels l'effet de "marches d'escalier" est éliminé.

Cet algorithme reprend l'idée de base de celui qui a été décrit par Bresenham (Bres65), et nous allons tout d'abord examiner ce dernier. Ces algorithmes peuvent s'appliquer à tout segment de droite, mais nous limiterons cette présentation au cas où l'origine est en (Xi,Yi) et l'extrémité en (Xi+Dx,Yi+Dy), avec Dx>0, Dy>0 et Dx>Dy, (Xi, Yi, Dx, Dy étant des entiers), sachant que l'on peut toujours se ramener à ce cas.

L'idée de base de l'algorithme de Bresenham, qui est la même pour la plupart des autres, peut se résumer de la manière suivante:

```
P.X := Xi; P.Y := Yi;
P.A(1):= Intensité;
 pour i:=1 à Dx faire debut
     Mettre(P, trame);
      si "point-trop-loin" alors P.Y := P.Y+1
                                 P.X := P.X+1
                                 fin
```

L'originalité de la méthode se trouve dans le façon dont elle procède au test "point-trop-loin". Ce test porte sur le successeur du point auquel on vient d'affecter une valeur, et on détermine si ce doit être (P.X+1,P.Y+1) ou (P.X+1,P.Y), car étant donné les hypothèses initiales, ce sont les deux seuls possibles. Le critère permettant de faire ce choix est simple. Soit D(X,Y) la fonction Y*Dx-X*Dy+C où C est une constante telle que tout point (X,Y) avec D(X,Y)=0 est un point de la droite portant le

trait que l'on échantillonne. On constate que si ABS(D(X,Y1)) est inférieure à ABS(D(X,Y2)) alors le point (X,Y1) est plus près de la droite que le point (X,Y2). Dans le cas présent, on doit comparer les positions de deux points dont l'un est (P.X+1,P.Y+1) et l'autre (P.X+1,P.Y). Soit la somme:

$$E = D(P.X+1,P.Y+1) + D(P.X+1,P.Y)$$

que l'on peut réecrire:

$$E = Dx - 2*Dy + 2*D(P.X,P.Y)$$

Si D(P.X+1,P.Y+1) et D(P.X+1,P.Y) sont tous deux négatifs, les deux points sont sous la droite, et il faut choisir (P.X+1,P.Y+1).

Si ils sont tous deux positifs, c'est alors (P.X+1,P.Y) qui doit être choisi.

Dans le cas où les deux membres sont de signes contraires, il faut choisir celui dont la valeur absolue est la moins élevée, donc celui dont le signe est opposé à celui de la somme. Donc dans tous les cas où E est négative, le successeur de (P.X,P.Y) est (P.X+1,P.Y+1) sinon c'est (P.X+1,P.Y). On remarque qu'il faut à tout moment conserver la puissance D(P.X,P.Y) du point qui vient d'être choisi, et ceci se fait aisément en utilisant celle du précédent. Initialement cette valeur est nulle puisque le point initial se trouve sur la droite. L'algorithme devient alors:

```
e:= Dx - 2*Dy ;
 pour i :=1 à Dx faire debut
            Mettre(P,trame);
             si e < 0 alors debut
                 y := y+1;
                 e := e + (2*Dx-2*Dy);
                          fin
              sinon e := e - 2*Dy;
             x := x+1;
                   fin
```

Comme pour tous les algorithmes en tout-ou-rien, les traits produits présentent des "marches d'escalier". Dans ce qui suit, nous présentons une méthode éliminant cet effet. Elle dérive de celle que nous venons de voir, mais au lieu d'affecter l'intensité du trait à un seul point, on l'affecte

à deux points dans des proportions fonction de leurs distances respectives à la droite.

Il faut tout d'abord partir de la constatation que le trait passe toujours entre deux points (X,Y) et (X,Y+1) (ou sur l'un des deux à la limite). Celui des deux qui est le plus proche de la droite sera appelé point principal c'est celui qui est selectionné par l'algorithme de Bresenham, l'autre sera appelé point secondaire. Notons a le point (P.X,P.Y), b le point (P.X,P.Y+2), c le point (P.X,P.Y+1), d le point (P.X+1,P.Y), et e le point (P.X+1,P.Y-1). comme indiqué sur la figure 16.

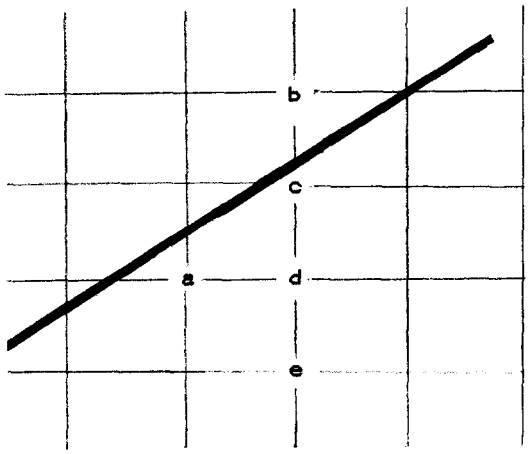


Figure 16.

Nous avons:

$$D(b) = D(a) - Dy + 2*Dx$$
$$D(c) = D(a) - Dy + Dx$$
$$D(d) = D(a) - Dy$$
$$D(e) = D(a) - Dy - Dx$$

Si D(c) est négatif alors, c est point principal, et b point secondaire.

Si D(d) est positif alors d est point principal, et e point secondaire.

Dans les autres cas, d ou c est point principal, ceci étant determiné de la même manière que dans l'algorithme précédent, et l'autre est point secondaire.

La répartition d'intensité entre les deux points retenus se fait dans le rapport inverse de celui de leurs puissances à la droite. L'algorithme est le suivant:

(Les couples <principal, secondaire> sont indiqués en fin de ligne lorsqu'ils sont determinés)

```
E:=0;
X:=Xi; Y:=Yi;
 pour i de 1 à Dx faire debut
E:=E-Dy; X:=X+1;
 si E ≥ 0 alors Yp:=Y-1;                                  <d,e>
     sinon si E+Dx ≤ 0 alors debut                        <c,b>
                              Y:=Y+1;
                              Yp:=Y+2;
                              E:=E+Dx;
                                   fin
                 sinon si 2*E+Dx ≤ 0 alors debut          <c,d>
                                        Y:=Y+1;
                                        Yp:=Y;
                                        E:=E+Dx;
                                             fin
                          sinon YP:=Y+1;                  <d,c>
P.X:=X;
P.Y:=Y;
P.A(1):=(1-ABS(E)/Dx)*Intensité;
Mettre(P,trame);
P.Y:=Yp;
P.A(1):=Intensité-P.A(1);
Mettre(P,trame);
                        fin
```

L'algorithme de Bresenham présente l'avantage de ne faire que des additions et soustractions entières ou multiplications par 2, ce qui n'est pas le cas de cette version modifiée, qui fait une règle de trois pour le calcul de l'attribut en un point.

La figure 17 montre un ensemble de vecteurs échantillonés à gauche par l'algorithme de Bresenham, et à droite par la version modifiée. On remarque que dans le second ensemble, les marches d'escalier ont été supprimées. Par contre, on voit apparaitre un effet de torsades, que l'on retrouve dans les vecteurs améliorés présentés dans d'autres publications (Cr78.2, BaFu79,..). Il faut noter que l'utilisation de cet algorithme sur un terminal vidéo produit des résultats similaires.

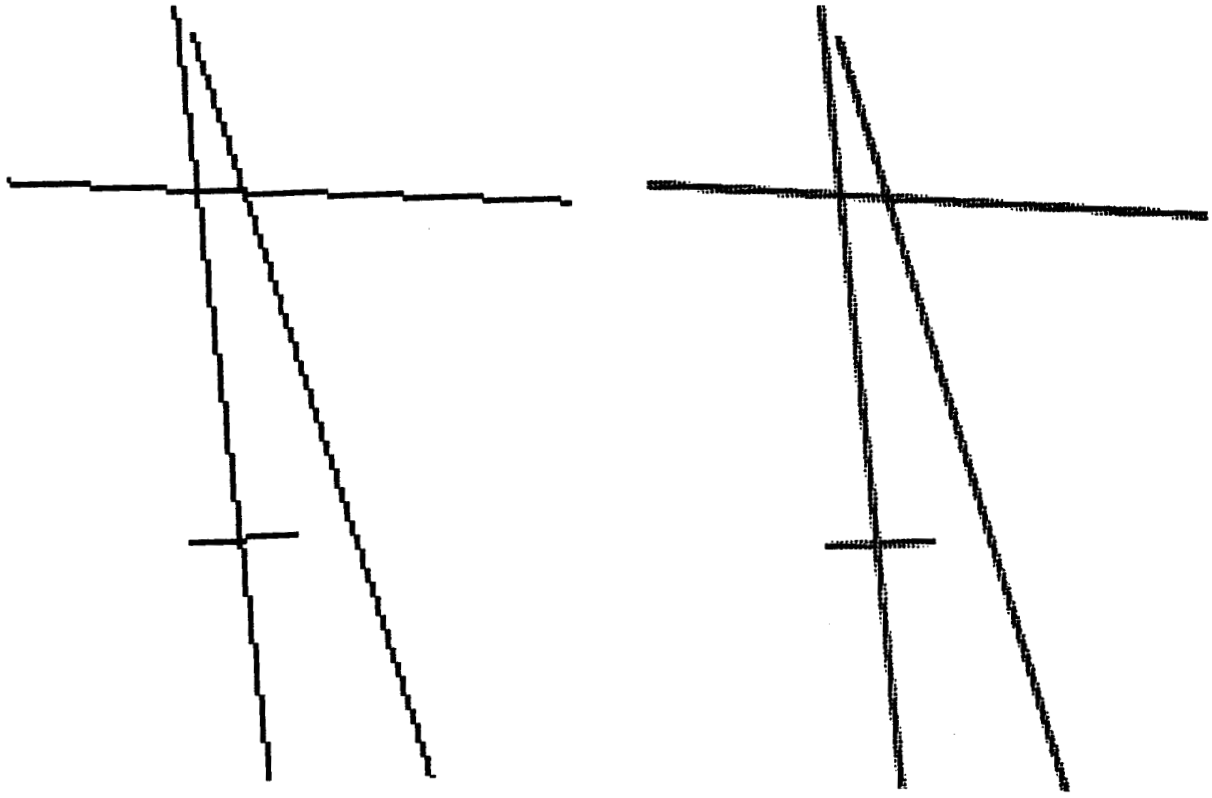


Figure 17.

ANNEXE 3: Le langage de commande du TEKTRONIX 4027.

Le Tektronix 4027 est un terminal de synthèse d'images simple dont la présentation permet de préciser certaines notions présentées dans le premier chapitre de cette thèse.

Nous n'examinerons qu'une partie de ses possibilités, mais le role complémentaire du synthétiseur et du convertisseur apparaitront clairement. Il recoit des commandes sous formes de chaines alphanuméériques dont la signification est précisée dans ce qui suit.

Le 4027 dispose d'une mémoire d'image, permettant éventuellement de mémoriser plus de pixels que ce qui est visualisable à un instant donné sur l'écran (*1).

Commandes au synthétiseur

On trouve tout d'abord une commande permettant de décrire la partie active de la mémoire d'image. Cet ordre s'ecrit:

```
GRA  Y1  Y2  ( X1  X2 )
```

Il indique que l'origine des coordonnées pour les primitives sera en (X1,Y1) et qu'il ne faudra visualiser que ce qui se trouvera dans le rectangle défini par (X1,Y1) et (X2,Y2)(*2). Cet ordre est équivalent à :

```
def-fenetre(1,1,X2-X1+1,Y2-Y1+1,T);
def-cloture(X1,Y1,X2,Y2,T);
```

où T est ignoré.

On trouve ensuite la commande d'initialisation:

---

*1 nous nous interessons à ce que perçoit l'utilisateur et non à ce qui se passe en réalité sous le capot, et donc le mode de gestion de cette mémoire ne sera pas abordé

*2 Nous supposerons que toutes les coordonnées fournies sont exprimées en nombre de pixels par rapport au coin bas gauche de l'écran, lorsqu'il s'agit de gestion d'écran

ERA G v

Elle permet de ranger la valeur v dans tous les pixels da la partie active de la mémoire d'image. Elle correspond donc à:

ini-tranche(1,v);

On trouve ensuite tout un ensemble de commandes permettant de décrire des primitives. Nous ne nous interesserons pas à celles qui concernent les caractères, mais uniquement à celles qui permettent de décrire des taches et des lignes.

Tout d'abord, on trouve des commandes permettant d'indiquer quelle sera la substance à utiliser pour les primitives qui vont être définies.

LIN n permet de choisir dans un catalogue prédéfini le style de tracé des vecteurs qui seront tracés.

COL c indique quelle sera la couleur employée pour les primitives. Eventuellement, ce pourra être une texture définie au préalable.

Les primitives sont définies par des commandes géométriques associées implicitement à la dernière substance définie. Il n'existe pas de domaine ou de transformation. Il existe plusieurs commandes géométriques, chacune correspondant à un couple (Classe, Interpretation)(*1) particulièr. Ce sont:

CIR ..... (Trait, cercle)

VEC ..... (Trait, polygone)

PIE ..... (Tache, cercle)

POL ..... (Tache, polygone)

Les pointillés figurent des suites de nombres entiers correspondant à des coordonnées ou des distances, suivant le contexte.

---

*1 Ces deux termes se comprennent aisément dans le contexte où nous les voyons, pour plus de détails, se reporter à (GGOu80).

On trouve par ailleurs principalement cinq commandes pour le convertisseur. Les trois premières concernent la modification de table de couleur, les deux autres correspondent à des déplacements. On notera qu'il n'existe qu'une seule image logique, correspondant à la seule tranche existante, et sur laquelle il n'est pas possible de faire le moindre traitement. De plus, cette image est constamment affichée, et il n'existe donc pas de commandes liées à la visualisation.

La table de couleur de ce terminal reçoit en entrée un pointeur entre 0 et 7 (3bits) et produit en sortie 3 fois 2 bits . On peut donc choisir huit couleurs parmi 64. L'utilisation de cette table n'est pas optionnelle, et donc il n'existe pas de commandes pour indiquer si on veut l'utiliser ou non. Les commandes de manipulation de la table permettent de modifier son contenu de trois manières différentes. Il y a tout d'abord:

MIX n R V B

R, V et B sont respectivement trois nombre indiquant la valeur de la composante rouge de la verte et de la bleue, exprimées en centièmes de valeur maximale d'intensité pour une composante. n indique l'élément de la table à changer. Cette commande est équivalente à:

def-couleur(n ,R, V, B);

Une autre commande est:

MAP n H L S

H, L et S sont trois nombres correspondant respectivement à une teinte, une brillance, et une saturation. Le schéma de la figure 18. (bien que noir et blanc) donne une indication de la manière dont ces valeurs sont interprétées pour choisir une couleur parmi les 64 disponibles. Cette commande est donc semblable à la précédente, le seule différence résidant sur le système de coordonnées utilisé pour désigner une couleur (voir à ce sujet (JoGr78) et (Sm78.1) ainsi que (SIBr79)). La dernière commande de manipulation de table de couleur n'affecte pas une valeur dans la table, mais modifie une valeur existante. Il s'agit de:

RMAP n H L S

H, L et S ne sont pas affectées au contenu de l'élément n de la table, mais lui sont ajoutées.

Les deux dernières commandes qui nous interressent sont:

RDO DY

RUP DY

Elles correspondent respectivement à:

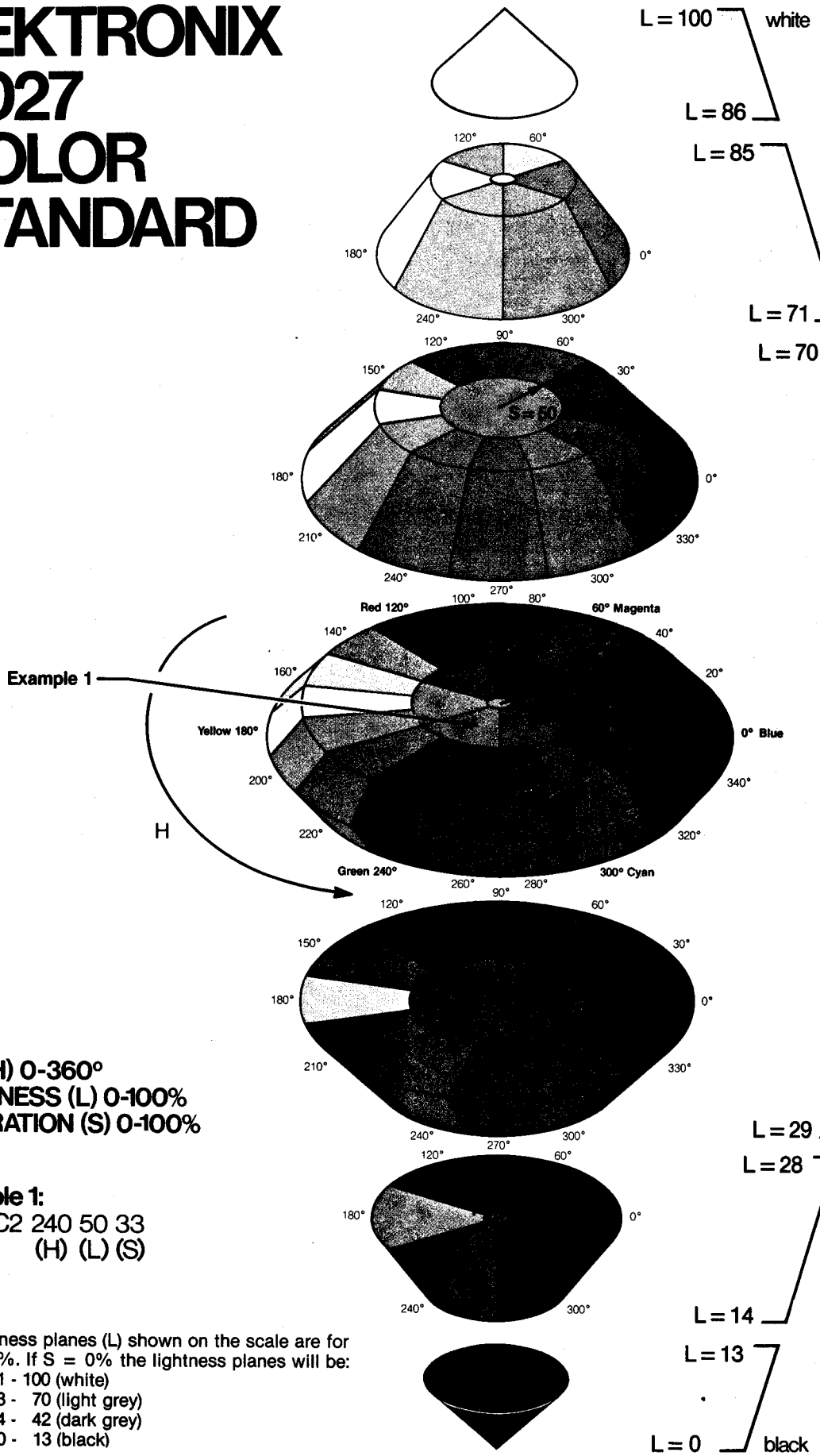
Deplacer (I1, I2, 0, -DY)

et

Deplacer (I1, I2, 0, DY)

Où les identificateurs d'images I1 et I2 ne sont pas utilisés.

TEKTRONIX 4027 COLOR STANDARD



HUE (H) 0-360°
LIGHTNESS (L) 0-100%
SATURATION (S) 0-100%

Example 1:
!MAP C2 240 50 33
(H) (L) (S)

***Note:**
The lightness planes (L) shown on the scale are for S = 100%. If S = 0% the lightness planes will be:
L = 71 - 100 (white)
L = 43 - 70 (light grey)
L = 14 - 42 (dark grey)
L = 0 - 13 (black)

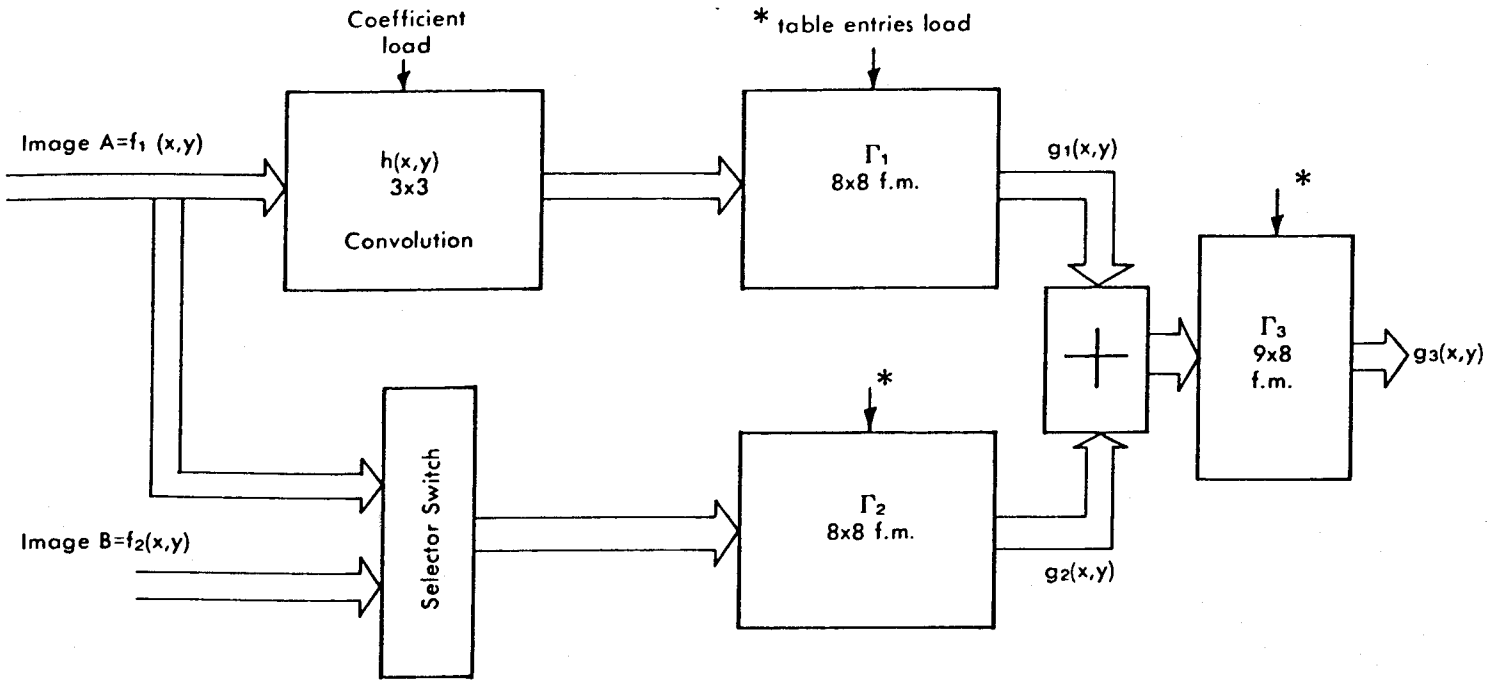
Figure 18. Description des couleurs pour le Tektronix 4027.

ANNEXE 3: L'architecture du COMTAL VISION One/20

COMTAL VISION One/20 est une machine qui a été développée pour des applications de traitements d'images, et son organisation est assez représentative de celle des autres matériels de la même gamme. On constate qu'elle est principalement composée d'une mémoire d'images, partagée par plusieurs convertisseurs pouvant gérer chacun leur propre écran.

Chaque convertisseur dispose d'opérateurs cablés sur des images tels que ceux qui ont été décrits dans le chapitre III, et on remarque que le fait que ces opérateurs soient cablés limite les possibilités de composition d'opérateurs entre eux (figure 19.). Les opérations sont effectuées en temps réel, c'est à dire 25 fois par seconde sur chaque pixel, et ceci donne une première idée des possibilités que l'on peut attendre des machines graphiques à venir.

La figure 20 donne un exemple de composition de plans de mémoire d'image et de convertisseurs. Elle permet entre autres de montrer la liberté dont on dispose pour bâtir une configuration, et ceci est également caractéristique de tous les matériels vendus à l'heure actuelle (Grinnel, Ramtek, Aydin,.....).



3x3 CONVOLUTION ± 8 BIT COEFFICIENTS

OPTIONS: 9 MULT & 8 ADDS/PIXEL => 133 M operations/sec. (MIPS)

$g_1(x,y)=\Gamma_1\{f_1(x,y)*h(x,y)\}$

$g_2(x,y)=\Gamma_2\{f_1(x,y) \text{ or } f_2(x,y)\}$

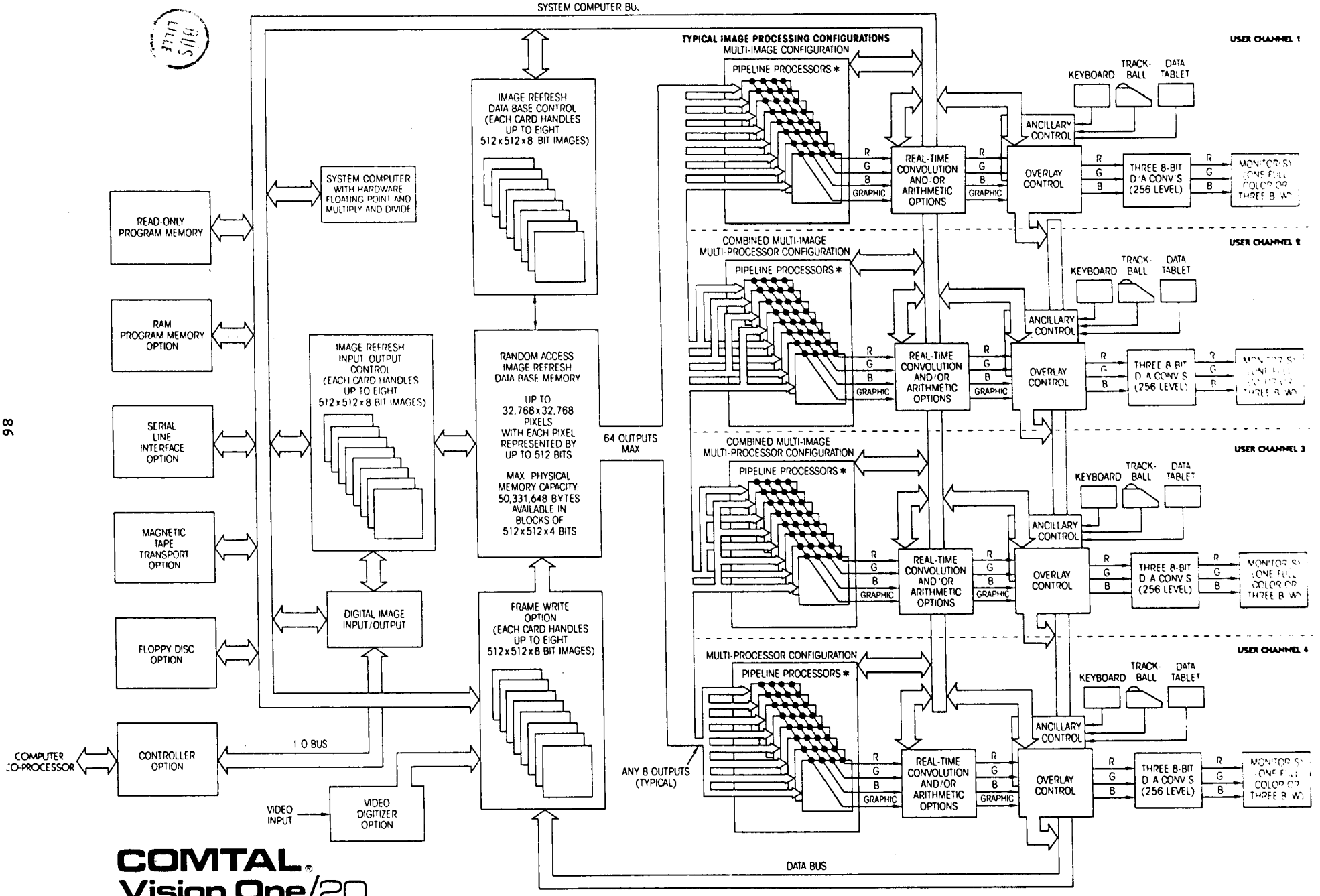
$g_3(x,y)=\Gamma_3\{g_1(x,y)+g_2(x,y)\}$

Schéma d'enchainement des fonctions de combinaison

Figure 19.

PAGE SUIVANTE:

Figure 20. Structure générale d'un système COMTAL Vision One/20



COMTAL® Vision One/20

SYSTEM BLOCK DIAGRAM

REFERENCES BIBLIOGRAPHIQUES

(AnTo78) ANDONIAN K.S., TOIDA S.
Multi-source Illumination and Shadowing.
Proc. Design Automation Conf. pp158-163. 1978.

(AoLe78) AOKI M., LEVINE M.D.
Computer Generation of Realistic Pictures.
Computer & Graphics . Vol 3. pp 149-161. 1978.

(Appe67) APPEL A.
The Notion of Quantitative Invisibility and the Machine Rendering of Solids.
Proceedings ACM National Meeting. pp387-393. 1967.

(Appe68) APPEL A.
Some Techniques for Shading Machine Rendering of Solids.
Proceedings SJCC. 1968. pp37-45.

(AsFu80) ASAMIZUYA N., FUTAI T.
Color Graphics and Graphic Animation by Minicomputer.
SMPTE Journal. Vol 89. pp721-724. October 1980.

(AtWG78) ATHERTON P.,WEILER K.,GREENBERG D.
Polygon Shadow Generation.
SIGGRAPH proceedings 1978.

(Baec79) BAECKER R.D.
Digital Video Displays Systems and Dynamic Graphics.
SIGGRAPH proceedings 1979.

(BaJo74) BARRET R.C., JORDAN B.W.
Scan Conversion Algorithms for a Cell-Organized Raster display.
Comm. ACM 17,3. March 1974.

(BaFu79) BARROS J.,FUCHS H.
Generating Smooth 2-D Monocolor Line-drawings on Video Displays.
SIGGRAPH proceedings 1979.

(BaTe78) BARROW H.G., TENENBAUM J.M.
Recovering Intrinsic Scene Characteristics from Images.
S.R.I. T.N.157. Menlo-Park. California. April 1978.

(BFSp77) BAUDELAIRE P., FLEGAL R.M., SPROULL R.F.
Spline Curve Techniques.
Xerox Palo-Alto Research Center. May 1977.

(BaSt80) BAUDELAIRE P., STONE M.
Techniques for Interactive Raster Graphics.
SIGGRAPH proceedings 1980. pp314-320.

(BeBa80) BECHTOLSHEIM A., BASKETT F.
High Performance Raster Graphics for Microcomputer Systems.
SIGGRAPH proceedings 1980. pp43-47.

(Bert77) BERTIN J.
La graphique et le traitement graphique de l'information.
Flammarion. 1977

(Blin77) BLINN J.F.
Models of Light Reflection for Computer Synthesized pictures.
SIGGRAPH proceedings 1977.

(Bl78.1) BLINN J.F.
Simulation of Wrinkled Surfaces.
SIGGRAPH proceedings 1978.

(Bl78.2) BLINN J.F.
Computer Display of Curved Surfaces.
University of Utah. PhD. 1978.

(BlNe76) BLINN J.F., NEWELL M.E.
Texture and Reflexion in Computer Synthesized Pictures.
Comm. ACM 19(10). pp542-547. Oct. 1976.

(BoKe70) BOUKNIGHT J., KELLEY K.
An Algorithm for Producing Half-Tone Computer Graphics Presentations with Shadows ans Movable Light Sources.
Proceedings AFIPS SJCC. Vol 36. pp2-10. 1970.

(Bres65) BRESENHAM J.E.
Algorithm for Computer Control of a Digital Plotter.
IBM System Jal. Vol4(1). pp25-30. 1965.

(BuTP75) BUI-TUONG PHONG.
Illumination for Computer Generated Images.
Comm. ACM 18,6. June 1975. pp 311-317.

(BuCr75) BUI TUONG PHONG, CROW F.
Improved Rendition of Polygonal Models of Curved Surfaces.
Proceedings 2nd USA-JAPAN Comp. Conf. Tokyo. August 1975.

(BCSW77) BURTNYCK N., CAREY P., STEELE K., WEIN M.
A Video Terminal for Interactive Graphics.
Proc. Fifth Man-Computer comm. conf. Calgary. May 1977.

(Catm74) CATMULL E.
A Subdivision Algorithm for Computer Display of Curved Surfaces.
University of Utah. Dept. of Computer Science. 1974.

(Catm75) CATMULL E.
Computer Display of Curved Surfaces.
Proc. Conf. on Computer Graphics, Pattern Recognition & Data Structures. IEEE. ppll-17. May 1975.

(Catm78) CATMULL E.
Hidden-Surface Algorithm with Anti-Aliasing.
SIGGRAPH proceedings. 1978.

(Catm79) CATMULL E.
A Tutorial on Compensation Tables.
SIGGRAPH proceedings 1979. ppl-7.

(CaSm80) CATMULL E., SMITH A.R.
3-D Transformations Of Images in Scan-Line order
SIGGRAPH proceedings 1980. pp279-285.

(Clar76) CLARK J.H.
Hierarchical Geometric Models for Visible Surface Algorithms.
Comm. ACM 17,1.pp32-42. January 1974.

(Clar79) CLARK J.H.
A Fast Algorithm for Rendering of Parametric Surfaces.
SIGGRAPH proceedings 1979. A paraitre dans Comm. ACM.

(Clar80) CLARK J.H.
A VLSI Geometry Processor for Graphics.
COMPUTER. pp59-68. July 1980.

(Crow76) CROW F.C.
The Aliasing Problem in Computer Synthesized shaded images.
Phd Thesis. University of Utah .Salt lake city. March 1976.

(Crow77) CROW F.C.
Shadow Algorithms for Computer Graphics.
SIGGRAPH proceedings 1977.

(Cr78-1) CROW F.C.
Shaded Computer Graphics in the Entertainment industry.
Computer. March 1978.

(Cr78-2) CROW F.C.
The Use of Grayscale for Improved Raster Display of Vectors and Characters.
SIGGRAPH proceedings 1978.

(Crow79) CROW F.C.
An Approach to Real-Time Scan Conversion.
AFIPS Conference proceedings. Vol48. pp157-163. 1979.

(Dung79) DUNGAN W.
A Terrain and Cloud Computer Image Generation Model.
SIGGRAPH proceedings 1979.

(Entw77) ENTWISLE J.
An Image Processing Approach to Computer Graphics.
Comp. & Graphics. Vol2. pp111-117. 1977.

(FaPr79) FAUX I.D., PRATT H.J.
Computational Geometry for Design and Manufacture.
ELLIS-HORWOOD Ld. Chihester G.B. 1979.

(FeGr80) FEIBUSH E., GREENBERG D.P.
Texture Rendering System for Architectural Design.
Computer-aided design. 12(2).pp67-71. March 1980.

(FLCo80) FEIBUSH E.A., LEVOY M., COOK R.L.
Synthetic Texturing Using Digital Filters.
SIGGRAPH proceedings 1980. pp294-301

(FoTD79) FOLEY J., TEMPLEMAN J., DASTYAR D.
Some Raster Extensions to the Core System.
SIGGRAPH proceedings 1979.

(FuLu78) FU K.S., LU S.Y.
Computer Generation of Textures Using a Syntactic Approach.
SIGGRAPH proceedings 1978. pp147-152.

(Fuch77) FUCHS H.
Distributing a Visible Surface Algorithm over Multiple Processors.
Proc. ACM 77. Seattle. October 1977. pp449-451.

(FuJo79) FUCHS H., JOHNSON B.W.
An expandable Multi-Processor Architecture for Video Graphics.
ACM SIGGARCH. Vol 7. pp58-67. April 1979.

(Gilo78) GILOI W.K.
Interactive Computer Graphics.
PRENTICE-HALL. 1978.

(GKS79) Graphical Kernel System.
Version 5.2. of DIN NI-5.9, November 1979.

(GoRi74) GORDON W.J., RIESENFELD R.F.
Bernstein-Bézier Methods for the Computer-Aided Design of Free-Form Curves and Surfaces.
Jal. of the ACM. Vol 21(2). pp293-310. April 1974.

(Gour71) GOURAUD H.
Continuous Shading of Curved Surfaces.
IEEE Trans. Comp. Vol C-20(6). June 1971. pp623-629.

(Grav76) GRAVE M.
Manipulation de surfaces en conception assistée par ordinateur.
IRIA. Rapport Laboria n210. Decembre 1976.

(Gr80.1) GRAVE M.
KISS: an Experiment in the Design of Image Synthesis Systems.
EUROGRAPHICS proceedings. 1980.

(Gr80.2) GRAVE M.
Etude d'un système graphique bati autour d'une mémoire d'image.
Congrès AFCET. Nancy. Novembre 1980.

(GrOu79) GRAVE M., OUANOUNOU G.
Basic Objects and Operators for Computer Animation.
EUROGRAPHICS proceedings. 1979.

(GGOu80) GRAVE M., GRONOFF J.D., OUANOUNOU G.
Etude d'opérateurs sur des objets graphiques élémentaires.
Congrès AFCET. Nancy. Novembre 1980.

(GSPC79) Graphics Standards Planning Committee (ACM/SIGGRAPH).
Second status report.
SIGGRAPH proceedings 1979.

(Gray79) GRAY R. & al.
Use of Standard TV Camera to Digitize Line Drawings.
Computer Application in Shipyard Operations and Ship Design III.
Kuo & al. Eds. IFIP, North-Holland. 1979.

(Hage77) HAGEN T. & al.
The Intermediate Language for Pictures.
Proceedings IFIP Congress. pp173-178. August 1977.

(Herb80) HERBISON-EVANS D.
Rapid Raster-Ellipsoid Shading.
Computer Graphics. Spring 1980.

(HeLi77) HERMAN G.T., LIU H.K.
A Simple Shading for Computer Displayed Surfaces.
TR-116. Univ. New-York at Buffalo. 1977.

(HoBa78) HORN B.K., BACHMAN B.L.
Using Synthetic Images to Register Real Images with Surface Models.
Comm. ACM. 21(11). November 1978.

(Horn75) HORN B.K.
Image Intensity Understanding.
M.I.T. A.I. memo335. August 1975.

(Jarv75) JARVIS J.F.
A Graphical Display System Utilizing Plasma Panels.
Computer & Graphics. Vol 1. pp175-180. 1975.

(JaJN76) JARVIS J.F., JUDICE C.N., NINKE W.H.
A Survey of Techniques for the Display of Continuous Tone Pictures on Bilevel Displays.
Comp. Grap. & Image proc. Vol 5. pp13-40. March 1976.

(JoGr78) JOBLOVE G.H., GREENBERG D.
Color Spaces for Computer Graphics.
SIGGRAPH proceedings 1978.

(Jone76) JONES B.
An Extended ALGOL-60 for Shaded Computer Graphics.
Computer Graphics Vol10 n1 Spring 1976.

(JoBa73) JORDAN B.W., BARRETT R.C.
A Scan-Conversion Algorithm with Reduced Storage Requirements.
Comm. ACM. 16,11. November 1973.

(KaSC75) KAJIYA J.T., SUTHERLAND I.E., CHEADLE E.
A Random-Access Video Frame-Buffer.
Proc. Conf on Computer Graphics, Pattern Recognition and Data Structures. IEEE. pp1-6. May 1975.

(KaGr79) KAPLAN M., GREENBERG D.P.
Parallel Processing Techniques for Hidden-Surface Algorithms.
SIGGRAPH proceedings 1979. pp300-307.

(Kawa72) KAWAI S. & al.
A Graphic System with Half-Tone and Area Coloring Capabilities.
Proceedings 1st USA-Japan Comp. Conf. pp320-324. October 1972.

(KaGr79) KAY D.J., GREENBERG D.
Transparency for Computer Synthesized Images.
SIGGRAPH proceedings 1979.

(Kend76) KENDER J.R.
Saturation, Hue and Normalized Color: Calculation, Digitalization Effects, and Use.
Carnegy-Mellon Univ. November 1976.

(KuWT74) KUNII T.L., WEYL S., TENENBAUM J.N.
A Relational Database Schema for Describing Complex Pictures with Color and Texture.
Proc. 2nd Int. Joint Conf. on Pattern Recognition. Lyngby-Copenhagen. August 1974.

(LPSt80) LAIB G., PUK R., STOWELL G.
Integrating Solid Image Capability into a General Purpose Calligraphic Graphics Package.
SIGGRAPH Proceedings. pp79-85. 1980.

(LaCa79) LANE J., CARPENTER L.
A Generalized Scan-Line Algorithm for the Computer Display of Parametrically Defined Surfaces.
Comp. Graphics & Image Processing. Vol 11. pp290-297. (1979).

(LCWB80) LANE J.M. & al.
Scan-line Methods for Displaying Parametrically Defined Surfaces.
Comm. ACM 23,1. January 1980. pp23-34.

(LeLM76) LEDUC-LEBALLEUR A., LUCAS M., MARTINEZ F.
GRIGRI: Logiciel de base pour l'utilisation des consoles de visualisation du C.I.C.G.
Note Technique 42. Université de Grenoble. Dec 1976.

(LeLM77) LEDUC-LEBALLEUR A., LUCAS M., MARTINEZ F.
Conception et réalisation d'un logiciel graphique interactif indépendant du contexte d'utilisation- Le logiciel de base: GRIGRI.
Revue RAIRO Informatique. pp147-173. 1978.

(Lema76) LEMAIRE A.
Contribution à la définition d'une norme en programmation graphique.
Note Interne. IRIA/Groupe graphique. 1976.

(Luca77) LUCAS M.
Contribution à l'étude des techniques de communication graphique avec un ordinateur. Eléments de base des logiciels graphiques interactifs.
Thèse d'état. Université de Grenoble. 1977.

(MaSh78) MALLGREN W.R., SHAW A.C.
Graphical Transformations and Hierarchic Picture Structures.
Comp. Grap. & Image proc. Vol 8. pp237-258. 1978.

(Max79) MAX N.L.
ATOMLLL:- ATOMS with Shading and Highlights.
SIGGRAPH Proceedings. 1979.

(Meri79) MERIAUX M.
Etude et réalisation d'un terminal graphique couleur tridimensionnel fonctionnant par taches.
These 3è cycle. Université de Lille. Janvier 1979.

(MeBo76) METCALFE R.M., BOGGS D.R.
Ethernet: Distributed Packet Switching for Local Computer Networks.
Comm. ACM. Vol 19(7). pp395-404. July 1976.

(Mund79) MUNDY J.L.
Visual Inspection of Metal Surfaces.
Proceedings AFIPS NCC. Vol48. pp227-231. 1979.

(Negr77) NEGROPONTE N.
Raster-Scan Approaches to Computer Graphics.
Computer & Graphics. Vol 2. pp179-193. 1977.

(NeNS72) NEWELL M.E., NEWELL R.G., SANCHA T.L.
A Solution to the Hidden-Surface Problem.
Proceedings ACM Nat. Conf. August 1972.

(Newe75) NEWELL M.E.
The Utilization of Procedure Models in Digital Image synthesis.
UTEC-Csc-76-218. Univ. Utah. Summer 1975.

(NeBl77) NEWELL M.E., BLINN J.F.
The Progression of Realism in Computer Generated Images.
Proc. ACM 77. Seattle. pp444-448. October 1977.

(NeSp74) NEWMAN W.M., SPROULL R.F.
An Approach to Graphics System Design.
Proceedings IEEE. Vol62. n4. April 1974.

(NeSp79) NEWMAN W.M., SPROULL R.F.
Principles of Interactive Computer Graphics.
Mc-GRAW-HILL. 2nd edition. 1979.

(Okos80) OKOSHI T.
Three Dimensional Displays.
Proceedings IEEE. Vol 68(5). pp548-564. May 1980.

(Park80) PARKE F.I.
Simulation and Expected Performance Analysis of Multiple Processor Z-Buffer Systems.
SIGGRAPH proceedings 1980. pp48-56

(Peuq79) PEUQUET D.J.
A Raster-Mode Algorithm for Interactive Modification of Line Drawing Data.
Comp. Grap. & Im. proc. Vol 10. pp142-158. 1979.

(PiWi80) PILLER E., WIDNER H.
Real-Time Raster Scan Unit with Improved Picture Quality.
Computer Graphics. Vol 14. (1-2). July 1980.

(Port78) PORTER T.K.
Spherical Shading.
SIGGRAPH Proceedings. 1978.

(Posd77) POSDAMER J.L.
Concurrent Processing for Computer Graphics.
Computer & Graphics. Vol 2. pp259-263. 1977.

(Prat78) PRATT W.K.
Digital Image Processing.
J. WILEY & SONS. 1978.

(Ries73) RIESENFELD R.F.
Application of B-Spline Approximation to Geometric Problems of Computer Aided Design.
UTEC-CsC-73-126. March 1973.

(RoAd76) ROGERS D.F., ADAMS J.A.
Mathematical Elements for Computer Graphics.
Mc-GRAW-HILL. New-york. 1976.

(RoKa76) ROZENFELD A., KAK A.C.
Digital Picture Processing.
ACADEMIC PRESS. 1976.

(RuWh80) RUBIN S.M., WHITTED T.
A 3-Dimensional RepResentation for Fast rendering of Complex Scenes.
SIGGRAPH proceedings. 1980.

(Salt79) SALTEL E.
Manuels FORTRAN 3D.
Notes internes INRIA groupe graphique. 1979.

(Sham78) SHAMOS M.I.
Computational Geometry.
Phd Thesis. Yale University. May 1978.

(Sher74) SHERR S.
The Technology and Characteristics of Computer Driven, Interactive, Graphic Display Systems.
Proceedings S.I.D. 15(1). 1974.

(Shin74) SHINN J.
Computer Generated TV Imagery.
S.I.D. Journal. 11(5). Sept/Oct 74.

(ShFl77) SHOHAT M., FLORENCE J.
Application of Digital Image Generation to the Shuttle Mission Simulator.
Proc. Summer Computer Simulation Conf. pp923-932. 1977.

(Shop74) SHOPIRO J.E.
Survey of Computer Graphics.
TR-1-A. Univ. Rochester. New-york. January 1974.

(Shou73) SHOUP R.G.
Some Quantization Effects in Digitally Generated Pictures.
S.I.D. Int. symp. pp58-59. 1973.

(Sh79.1) SHOUP R.G.
Some Experiments in Television Graphics and Animation Using a Digital Image Memory.
Digital Video, Vol 2, SMPTE, Scardsdale, New-york. 1979.

(Sh79.2) SHOUP R.G.
Color Table Animation.
SIGGRAPH proceedings 1979. pp8-13.

(SlBr79) SLOAN K.R., BROWN C.M.
Color Map Techniques.
Comp. Graphics & Image proc. Vol 10. pp297-317. Oct 1979.

(Slot76) SLOTTOW H.G.
Plasma Displays.
IEEE Trans. Electron. Devices. ED-27, 7. July 1976.

(Sm78.1) SMITH A.R.
Color Gamut Transform Pairs.
SIGGRAPH proceedings 1978.

(Sm78.2) SMITH A.R.
PAINT.
Technical Memo. n.7. NYIT. Old Westbury, NY 11568. July 1978.

(StOg75) STAUDHAMMER J., OGDEN D.J.
Computer Graphics for Half-Tone Three Dimensionnal Object Images.
Computer & Graphics. Vol 1. pp109-114. 1975.

(Ster79) STERN G.
Softcel: An Application of Raster Scan Graphics to Conventional Cel Animation.
SIGGRAPH Proceedings. 1979.

(SSSc74) SUTHERLAND I.E., SPROULL R.F., SCHUMACKER R.A.
A Characterization of Ten Hidden-Surfaces Algorithms.
ACM computing surveys 6(1). March 1974.

(SuHo74) SUTHERLAND I.E., HODGMAN G.W.
Reentrant Polygon Clipping.
Comm. ACM. 17(1). January 1974.

(Tani78) TANIMOTO S.L.
Improving Interactions with a Raster Display.
Computer Graphics. Vol 11. n.4. Mar 1978.

(TeFB80) TENENBAUM J.M., FISHLER M.A., BARROW H.G.
Scene Modeling: A Structural Basis for Image Description.
Comp. Graphics & Image proc. Vol 12. pp407-425.April 1980.

(Vill74) VILLALOBOS L.
Virtual Graphic Device.
Computer Graphics. Winter 1974.

(Watk70) WATKINS G.S.
A Real-Time Visible Surface Algorithm.
Univ. Utah. UTEC-CSc-70-101. June 1970.

(Warn79) WARNER J.R.
DIGRAF: User's Guide.
University of colorado. Boulder. January 1979.

(Warn69) WARNOCK J.E.
A Hidden-Surface Algorithm for Computer Generated Half-Tone Pictures.
Univ. Utah. TR4-15. 1969.

(WeAt77) WEILER K., ATHERTON P.
Hidden Surface Removal Using Polygon Area Sorting.
SIGGRAPH proceedings. pp214 222. 1977.

(Whit79) WHITTED T.
An Improved Illumination Model for Shaded Displays.
SIGGRAPH proceedings 1979. A paraitre dans CACM.

(Wi-L78) WILLIAMS J.
Casting Curved Shadows on Curved Surfaces.
SIGGRAPH proceedings. 1978.

(Wi-R78) WILLIAMS R.
Image Processing and Computer Graphics.
IBM Research Lab. RJ2336(31468). November 1978.

CLASSIFICATION DES REFERENCES BIBLIOGRAPHIQUES

Textes généraux

(Luca77), (NeSp79), (Negr77), (Shop74), (Gilo78), (NeBl77)

Visibilité, échantillonnage, (Anti-)aliasing

(Appe68), (Bafu79), (Catm74), (Catm78), (Clar79), (Crow76), (Cr78-2), (Crow78), (FLCo80), (KaGr79), (LCWB80), (NNSA72), (Shou73), (SSSc74), (Watk70), (Warn69), (WeAt77), (APPe67), (Catm75), (Laca79), (Bl78.2)

Traitements d'images, analyse de scènes

(BaTe78), (Horn75), (Kend76), (Prat78), (RoKa76), (TeFB80), (Entw77), (TeFB80)

Logiciels graphiques généraux, extensions de (..)

(FoTD79), (GKS79), (GSPC79), (Hage77), (Jone76), (LPSt80), (LeLM76), (LeLM77), (Salt79), (Gr80.2), (Warn79)

Ombrages

(Blin77), (BuTP75), (Gour71), (Herb80), (HeLi77), (HoBa78), (Max79), (Whit79), (Port78)

Textures

(BlNe76), (FeGr80), (FLCo80), (FuLu78), (Bl78.1), (Bl78.2)

Ombres portées

(AtWG78), (Crow77), (AnTo78), (BoKe70), (Wi-L78)

Structures de systèmes, systèmes spécifiques

(Baec79), (BCSW77), (Clar80), (Gr80.1), (Meri79), (NeSp74), (Park80), (PiWi80), (Sh79.1), (Kawa72), (StOg75), (KaSc75), (ShFl77)

Modélisation et description de scènes

(Clar76), (KuWT74), (MaSh78), (Newe75), (RuWh80), (TeFB80)

Techniques diverses et applications

(BFSp77), (BaSt80), (Bert77), (Catm80), (Cr78-1), (Gray79), (GrOu79), (GGOu80), (JaJN76), (Peuq79), (Sh79.2), (SlBr79), (Tani78), (Ster79)

Géométrie, modèles géométriques.

(FaPr79), (Sham78), (RoAd76), (Ries73), (GoRi74)

Parallelisme

(Posd77), (FuJo79), (KaGr79), (Park80), (Fuch77)

REFERENCES MATERIELLES

(*Bens) BENSON : logiciels élémentaires de pilotage de traceurs

(*Comt) COMTAL VISION One/20

(*HP14) HEWLETT-PACKARD 9845

(*Matr) MATRIX

(*Orio) ORION

(*Prin) PRINCETON 8500M

(*Prtr) PRINTRONIX

(*TK14) TEKTRONIX 4014

(*TK27) TEKTRONIX 4027

(*Vers) VERSATEC