THESE

présentée à l'U.F.R. de

Mathématique et Informatique

de l'Université Louis Pasteur

pour obtenir le titre de

DOCTEUR DE L'UNIVERSITE

LOUIS PASTEUR DE STRASBOURG

mention Sciences

spécialité Informatique

par

Raymond RIPP

Animation graphique et interactivité

---

soutenue le 23 mai 1991 devant la commission d'examen:

MM.	J.F. DUFOURD	Président, Rapporteur interne
	J. FRANÇON	Directeur
	B. PÉROCHE	Rapporteur externe
	J.C. SPEHNER	Examinateur

Sommaire

Chapitre 1

Introduction

L'**image de synthèse** est devenue un élément important dans la communication entre l'homme et la machine; par son intermédiaire, l'ordinateur peut nous fournir les résultats de ses calculs sous une forme plus facile à analyser que des tableaux de chiffres, que ce soit par des graphiques bidimensionnels ou des images plus ou moins réalistes d'objets tridimensionnels réels ou imaginaires [Bertin 1977].

L'augmentation des puissances de calcul permet maintenant l'**animation**; l'écran affiche plus qu'une photographie: il s'y déroule un film. Des phénomènes complexes peuvent être étudiés en fonction du temps, une dimension de plus est offerte à nos yeux.

L'ordinateur sera désormais capable de modifier l'affichage, en temps réel, en réponse à des commandes. L'utilisateur agit directement sur l'image, il donne des ordres à la machine en manipulant les objets graphiques. L'écran est devenu un moyen d'échange d'informations entre l'homme et la machine [Foley et van Dam, 1982].

Pour que l'**animation graphique interactive** soit réellement utilisée, il faut qu'elle soit accessible à tous, même au non-informaticien. L'échange entre l'utilisateur et la machine ne peut se faire que si sa mise en œuvre est simple. Il doit se substituer aux échanges de commandes par clavier-écran. L'immense succès des interfaces souris-écran tels que les proposent maintenant la plupart des ordinateurs est là pour en témoigner [Foley et van Dam, 1982].

Le simple déplacement d'un dessin sur un écran ou la modification de l'objet graphique tridimensionnel représentant un corps humain en mouvement ou un visage

qui parle, sont des exemples d'animation graphique. Les techniques utilisées peuvent être plus ou moins sophistiquées; elles dépendent évidemment de la nature de l'application mais aussi dans une large mesure du matériel utilisé. S'il est possible aujourd'hui de réaliser des films de synthèse d'un réalisme parfait, les temps de calcul nécessaires sont, dans la plupart des cas, loin de permettre le temps réel et l'interactivité; cela n'est concevable, à l'heure actuelle, que si le graphisme et les techniques d'animation sont relativement simples.

C'est dans ce but que nous avons développé un logiciel, *A*GRAPH, qui permet la création et la manipulation d'objets graphiques animés ainsi que la réalisation de séquences d'animation qui peuvent éventuellement être filmées par une caméra. Nous nous restreignons au cas où les objets graphiques sont composés de parties rigides auxquelles peuvent être appliquées des transformations géométriques telles que translations, rotations, homotéthies et des modifications de couleur et de luminosité. La mise à jour de ces paramètres peut alors être assez rapide pour permettre l'animation en temps réel et la répercussion immédiate à l'écran des commandes interactives de l'utilisateur. La suite des valeurs nécessaires à l'animation est calculée par **interpolation de positions clés** définies interactivement par l'utilisateur.

Nous avons étudié, conçu et mis au point des programmes d'interpolation pour obtenir, en particulier, des mouvements continus et lisses, tenant compte du temps, c'est-à-dire des durées entre les positions clés successives. Nous avons plus particulièrement porté notre attention sur l'interpolation des orientations modélisées par des quaternions. Afin de pouvoir juger des résultats, des outils de visualisation graphiques et de représentations animés des paramètres interpolés ont été développés.

*A*GRAPH est aussi et surtout un logiciel qui permet à un non-informaticien de créer, sans programmation, une séquence d'animation. Il intègre tous les outils nécessaires à la réalisation d'un film, c'est-à-dire les aspects *mise en scène* pour la définition (et redéfinition) des positions clés, le calcul des interpolations, le montage des séquences, leur visualisation en temps réel, autorisant, entre autres, de nouvelles opérations de mise en scène, et finalement les étapes de *prise de vues* par caméra, en vitesse normale ou vue par vue.

Les applications que nous avons développées ont été implantées sur une station graphique *PS300* Evans & Sutherland, machine très performante qui permet ef-

fectivement l'animation en temps réel d'objet graphiques filaires complexes composés de plus de 50000 vecteurs. Son mode de fonctionnement, particulièrement intéressant, est basé sur la notion de **cadencement par les données**. L'animation est obtenue par mise à jour d'une structure de données graphique par l'intermédiaire d'un réseau de fonctions cadencé par les données.

Programmer la station graphique consiste, d'une part, à définir une structure de données graphique et d'autre part à construire le réseau de fonctions qui doit la modifier. Ce langage de programmation, *data flow*, spécialement conçu pour la mise au point d'un **système interactif** est par ailleurs particulièrement bien adapté à l'expression des applications tournant sur les ordinateurs **parallèles** de demain. Ce nouveau type de langage nous a conduit à réapprendre à programmer, à inventer et créer les méthodes et outils indispensables à la mise au point de gros programmes. Le cadencement par les données s'applique très bien à l'animation et offre une solution élégante qui mérite des développements.

La station graphique *PS300* Evans & Sutherland nous a amenés à voir le graphisme et l'animation sous un angle nouveau, tant par sa **puissance** d'affichage que par l'**originalité** de son mode de fonctionnement ou de sa programmation: les applications auxquelles notre travail a donné lieu sont nombreuses et variées.

Après une présentation générale de l'*animation graphique par ordinateur* nous exposons au chapitre 2 les concepts et solutions que nous avons retenus pour *A*GRAPH. Nous décrivons alors globalement le mode de fonctionnement du logiciel afin de comprendre son architecture et de permettre son utilisation.

Au chapitre 3 nous abordons les problèmes de programmation des systèmes interactifs, par un rapide survol des langages de programmation du parallélisme puis par une étude du cadencement par les données, base de la programmation de la station graphique qui est décrite en détail en annexe A. La partie interactive de *A*GRAPH est présentée dans ce chapitre; une description succinte de tous les modules composant le réseau de fonctions cadencés par les données est faite dans l'annexe B, le module de commande de la caméra est détaillé dans l'annexe C.

Les chapitres 4 et 5 traitent des problèmes d'interpolation et d'utilisation des quaternions pour le calcul des orientations. Les solutions retenues pour *A*GRAPH y

sont présentées. Un organigramme général du logiciel implanté sur l'ordinateur hôte est donné en annexe D.

Enfin, le chapitre 6 nous expose les applications.

Chapitre 2

Synthèse d'images, animation graphique

Dans toute application de conception assistée par ordinateur la représentation des résultats et la prise en compte des ordres de l'utilisateur sont primordiales. De ce fait, CAO et synthèse d'images sont étroitement liées; en effet, l'image est souvent le **support** qui permet l'échange d'informations, dans les **deux sens**, de l'utilisateur vers l'ordinateur, de l'ordinateur vers l'utilisateur.

L'interface homme-machine, base de toute application de CAO, doit être plus conviviale que le difficile *dialogue* constitué des seuls échanges de commandes par l'intermédiaire du clavier et de l'écran alphanumérique, elle nécessite l'**animation** et l'**interactivité**.

2.1 L'échange d'informations

Nous percevons le monde qui nous entoure par l'intermédiaire de nos sens; pour que l'ordinateur puisse nous communiquer des informations, il doit faire appel à notre *vue* par l'image et le texte, à notre *ouïe* par synthèse de la parole et bientôt à notre sens du *toucher* [Foley 1987].

Réciproquement, il est possible de commander la machine par la voix, et surtout par nos gestes, que ce soit par l'intermédiaire du clavier (moyen classique et assez barbare qui est néanmoins la manière la plus simple d'entrer une commande)

ou par les mouvements de la main ou d'autres parties de notre corps (tête, yeux, bras, etc.) [Ware et Jessome, 1988].

Ainsi l'interface *souris-écran* est devenue un outil irremplaçable. Les déplacements de notre bras guident la souris, l'image, en plus de son contenu informationnel, nous sert de *support* pour indiquer à la machine *où* et *sur quoi* doivent être appliquées les opérations choisies.

Les simulateurs de vol illustrent parfaitement ces échanges d'informations. Les mouvements du pilote transmis par une multitude de capteurs (manche à balai, pédales, boutons et clés, position de la tête et des yeux etc.) sont pris en compte par l'ordinateur qui calcule les mouvements de l'avion virtuel et retransmet au pilote les informations adéquates. Citons, entre autres, la visualisation du paysage, l'affichage des indicateurs mais aussi l'inclinaison et la mise en vibration de la plate-forme de simulation ou la pressurisation de la combinaison anti-G afin de simuler les accélérations perceptibles par le sens du toucher, ainsi que les forces de rétroaction à appliquer sur le manche à balai.

2.2 L'image de synthèse

L'utilisation d'un écran graphique est de plus en plus **répandue**; cela est dû, bien sûr, à la baisse des coûts de ce type de materiel mais surtout au confort que cela apporte à l'utilisateur. S'il est maintenant (28 ans après I.E. Sutherland, [Sutherland 1963]) devenu évident que se servir d'un ordinateur ne peut se faire que par l'intermédiaire d'un écran, l'interprétation des résultats est souvent plus aisée par une **visualisation graphique**. Comment analyser un phénomène sans en tracer la courbe d'évolution en fonction du temps, comment présenter des résultats statistiques sans les représenter par des histogrammes? Quel serait le travail du cristallographe face à la structure tridimensionnelle d'une molécule composée de plusieurs centaines d'atomes s'il ne pouvait la *voir* que par une suite de chiffres sur un listing? Que ferait le mécanicien ou l'architecte sans le support graphique que représentent les plans de construction?

L'image fournie par l'ordinateur doit être interprétable et **compréhensible** facilement; ce n'est déjà pas simple pour des graphiques bidimensionnels ou des cartes

géographiques [Bertin 1977], mais devient délicat -et parfois hypothétique- lorsqu'il s'agit de visualiser des objets tridimensionnels complexes ou de représenter des entités abstraites.

Le temps nécessaire au calcul d'une image peut varier de quelques fractions de seconde à plusieurs heures, ou jours ... Cela dépend beaucoup de la puissance de la machine mais surtout du type des images créées, celles-ci ne sont composées que de quelques ordres de tracé 2D, celles-là sont une représentation souvent trés **réaliste** d'objets tridimensionnels complexes [Sciences & Techniques, 1984]. Elles ne sont pas créées dans le même but, certaines ne sont qu'une façon simple et rapide de présenter un résultat, d'autres sont de véritables œuvres d'art.

2.3 Animation graphique

"Un dessin vaut mieux qu'un long discours...", un film est plus qu'une suite d'images.

Lorsque la puissance de la machine le permet, les images créées par l'ordinateur peuvent s'animer, et donner à l'utilisateur l'impression de voir se dérouler un film à l'écran.

Comme pour l'image fixe, les techniques sont nombreuses et dépendent à la fois des possibilités graphiques de la station de travail et du type d'animation souhaité.

Certaines animations sont destinées au vrai cinéma, faites par des professionnels avec les moyens des super-productions, d'autres ne sont, par exemple, que la suite des images stylisées d'un jeu vidéo sur un micro-ordinateur.

Certains ordinateurs permettent l'affichage en temps réel du film sur l'écran graphique, d'autres nécessitent la création des images, une à une, sur un support vidéo ou pellicule cinéma, pour leur visualisation par les procédés classiques du cinéma.

2.3.1 L'animation au cinéma et à la télévision

Les dessins animés de qualité, produits par exemple par les studios Walt Disney, sont dans une large mesure encore effectués manuellement; l'aide apportée par l'informatique est plus du domaine de l'organisation et de la gestion de l'énorme

quantité de documents (dessins propres à chaque personnage, décors, bande son, etc.) que de la création des images proprement dite [Moissinac 1984]. Les scénarios et dessins clés sont réalisés en 2D, avec du papier et un crayon, par des artistes dessinateurs, les images intermédiaires et les coloriages par des *"assistants animateurs qui donnent au mouvement une partie de son expression, car il ne s'agit pas d'un simple intervallage mécanique comme beaucoup d'informaticiens l'ont espéré"* (extrait de la thèse de J.C. Moissinac). Les problèmes liés à une automatisation possible de ce type d'animation sont encore nombreux, qu'ils soient techniques, pour ce qui concerne la qualité du résultat, humains, pour les changements de méthodes de travail que cela suppose, ou économiques, les coûts d'un traitement informatique étant encore nettement supérieurs à des productions classiques.

Certaines sociétés de productions cinématiques se sont spécialisées dans l'utilisation d'images de synthèse pour la réalisation des effets spéciaux de films tels que *Star Wars* ou *Tron* [Colonna 1987]. Il s'agit là, souvent, de créer des décors (espaces intersidéraux, vaisseau spatial) ou d'utiliser des techniques particulières de synthèse d'images pour simuler par exemple l'explosion d'un avion *(Starfighter)* ou les turbulences de l'atmosphère de Jupiter (*2010 Odyssée II*, [Odyssée 1985]). L'animation proprement dite n'est pas très utilisée, les séquences réalisées sont courtes et demandent généralement beaucoup de calculs et des mises au point spécifiques [Tyler 1984, State of the Art 1985].

Depuis quelques années les chaînes de télévision proposent des séries de dessins animés pour enfant d'une qualité souvent médiocre. Devant être produits vite et bon marché, ils utilisent des techniques très rudimentaires, à la fois dans la réalisation des images clés qui doivent être simples et dans la création des images intermédiaires qui sont calculés par des interpolations linéaires donnant aux mouvements cet aspect saccadé et non naturel *(Ulysse 31, Goldorak, etc.)* [Sciences & Techniques, 1984].

Comme pour les effets spéciaux, les films correspondants aux logos, génériques d'émission télévisées ou spots publicitaires sont réalisés par des équipes de professionnels du cinéma et d'informaticiens disposant de gros moyens de calcul et travaillant spécialement sur un scénario donné. Les stations graphiques utilisées sont souvent des machines 3D de hautes performances.

2.3.2 L'animation par ordinateur

Nous nous intéressons maintenant aux techniques et aux logiciels permettant de réaliser des séquences d'animation qui ne soient pas spécifiques et axées sur la création d'une seule séquence dont le prix de revient peut être illimité comme pour les films grand public ou les génériques de télévision. Ces logiciels peuvent être considérés comme des outils [Getto et Breen, 1990] mis à la disposition d'un utilisateur même non informaticien.

Les fonctionnalités d'un logiciel d'animation

Un logiciel d'animation devrait

- permettre à l'utilisateur de **créer des objets** graphiques animables,
- lui donner les moyens de les **animer**, que ce soit directement, par l'intermédiaire d'une interface interactive, ou indirectement, à l'aide de commandes plus ou moins explicites telles que *"tourner X de 10°"*, *"plier le genou"*, ou même *"Humphrey prend la main de Marylin..."* (dans *Rendez-vous à Montréal* [Magnenat-Thalmann
- faciliter l'**écriture de scénarios**, c'est-à-dire doit pouvoir accepter des suites de commandes ou même les créer lui-même,
- enfin, **visualiser**, sur l'écran graphique, le résultat, image par image ou à vitesse réelle si les temps de calcul et la puissance de la machine le permettent.

Toutes sortes de logiciels ont été mis au point [Lucas 1970], de simples programmes, écrits en un langage conventionnel appelant les routines de la librairie spécifique d'une station graphique donnée, aux applications d'Intelligence Artificielle où l'utilisateur *metteur en scène* dirige des acteurs *intelligents* doués de sentiments [Calvert 1988, Delesalle *et al.*, 1988].

Les logiciels existants

La modélisation d'un objet graphique en vue de son animation est autrement plus ardue que pour l'obtention d'une image seule où cela importe peu (...pourvu qu'on ait l'image). Le modèle doit être décomposé pour accepter facilement les mises

à jour de paramètres qui permettront l'animation sans pour cela réinventer l'objet ou le reprogrammer entièrement. La simulation d'un feu [Reeves 83], par exemple, ne se traite pas de la même façon que les mouvements de la main formée d'un grand nombre d'os et manipulés par des dizaines de muscles [Magnenat-Thalmann *et al.*, 1988].

Un objet graphique animable peut être une entité rigide, une construction hiérarchique arborescente dont les noeuds sont des transformations géométriques simples ou sophistiquées [Hanrahan et Sturman, 1985] ou même un programme complexe attendant des données [Hedelman 1984]. Cette distinction entre structure de données passive que l'on modifie, plus ou moins facilement, et programme actif qui attend de nouveaux paramètres pour s'exécuter, se retrouve dans les différents modèles de systèmes d'animation existants. On remarquera qu'elle est complètement masquée par la programmation *orientée objet* où ces deux notions sont en fait confondues.

Comme dans [Magnenat-Thalmann et Thalmann, 1985/a] et [Péroche *et al.*, 1988], nous répartissons les différents logiciels en:

- Systèmes à base de **positions clés** (nous y revenons en détail plus loin) où, souvent, l'utilisateur positionne interactivement des acteurs (relativement peu compliqués), l'ordinateur se chargeant de calculer les images intermédiaires par interpolation (*BBop*, [Stern 1983] et *A*GRAPH, [Amerein et Ripp, 1986]).

- Systèmes demandant une **programmation** de la part de l'utilisateur, que ce soit en des langages classiques comme *Fortran*, *C*, en liaison avec les routines graphiques comme *GKS* ou *Phigs*, ou par des moyens spécifiques à l'animation, permettant de créer simplement de vrais scénarios tels que *Gramps* [O'Donnel et Olson, 1981], *ASAS* [Reynolds 1982], *ASA++* [Gançarski 1988]. Ces programmes sont souvent écrit dans des **langages orientés objet** ou fonctionnant comme tel; la plus grosse réalisation semblant être le logiciel *The Clockworks* (100 000 lignes de *C* [Getto et Breen, 1990]).

- Plutôt que d'obliger l'utilisateur à donner explicitement les commandes directes créant le mouvement, certains systèmes se chargent de calculer les actions à effectuer soit par des méthodes de **cinématique inverse** [Badler 1987, Badler *et al.*, 1987] ou obéissant aux **lois de la dynamique** comme par exemple la gravité ou l'inertie des masses [Brotman 1988].

- L'apport des techniques d'IA et des systèmes experts permet, ou permettra, de faciliter le travail de l'utilisateur metteur en scène, la machine *acteur* ayant *appris* à jouer.

2.4 *A*Graph: un logiciel d'animation graphique

La station graphique *PS300* Evans & Sutherland permet l'animation temps réel d'objets graphiques 3D complexes dans la mesure où ils sont construits comme une arborescence dont les nœuds sont des transformations géométriques élémentaires telles que des rotations, des translations et des homothéties ou des modifications de couleurs et de luminosités. Elle offre des possibilités d'interactivité très performantes et autorise la visualisation en temps réel.

Pour offrir à l'utilisateur non informaticien un outil de réalisation de séquences d'animation nous avons choisi de développer un système d'animation par **interpolation de positions clés** que l'utilisateur définit **interactivement** au moyen des boutons et touches de fonction de la station graphique.

Le mode de fonctionnement très particulier de la machine, le **cadencement par les données**, que nous étudierons en détail par la suite, est parfaitement adapté à ce système d'animation. Toute la partie *mise en scène* et *animation* est gérée par les réseaux de fonctions cadencés par les données. Le calcul des interpolations est fait sur l'ordinateur hôte; il serait très intéressant de l'implanter sur ces réseaux de fonctions, autorisant à terme, si la puissance du (ou des) processeur(s) le permettait, le calcul de ces interpolations en "temps réel" (ou presque).

2.4.1 Utilisation du logiciel

L'utilisateur est d'abord **metteur en scène**: il dirige interactivement les acteurs pour définir les scènes-clés au moyen des boutons de la station graphique.

Il est aussi **cameraman**: il choisit pour chaque scène-clé la position, l'orientation et le réglage de la caméra virtuelle qui la filmera.

L'utilisateur est ensuite **intervalliste**: tous les paramètres des scènes ainsi définies sont enregistrés, sur demande du metteur en scène, par l'ordinateur hôte pour y être interpolés.

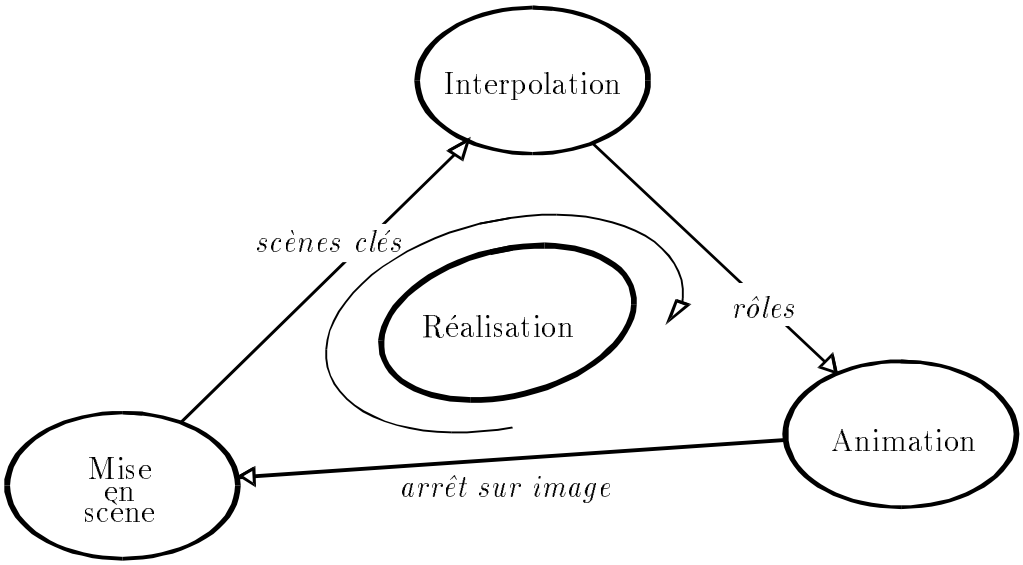


Figure 2.1: Schéma général de *A*GRAPH

L'utilisateur est finalement **spectateur**: la séquence d'animation se déroule sous ses yeux, en temps réel. Il peut à tout moment l'arrêter et refaire une mise en scène en modifiant interactivement l'un des acteurs.

La figure 2.1 montre les différentes étapes conduisant à la **réalisation** d'une séquence d'animation.

Le film qui se déroule en temps réel à l'écran peut éventuellement être **enregistré par une caméra cinéma ou un magnétoscope**.

2.4.2 Principe de fonctionnement

Les acteurs, la scène

Le terme *acteur* a en informatique plusieurs significations, l'une, d'après Hewitt [Hewitt *et al.*, 1973], désignant un processus indépendant communiquant avec d'autres acteurs par envoi de messages (c'est par exemple un *objet* dans un langage orienté objet), l'autre, défini dans [Reynolds 1982] est associé à un objet graphique animable.

Comme [Magnenat-Thalmann et Thalmann, 1985/b], nous adoptons la définition suivante:

Nous appellerons **acteur** un objet graphique défini par le quintuplet $A = (p, o, t, a, G)$,

où

- $p \in {}^3$ est sa position dans l'espace
- o son orientation (axe et angle de rotation (ou quaternion))
- $t \in {}^+$ son facteur d'échelle
- $a = (c, s, k)$ l'aspect défini par
  - la couleur $c \in [0, 2\pi]$
  - la saturation $s \in [0, 1]$
  - la valeur de contraste $k \in [0, 1]$
- G est l'ensemble des objets graphiques auxquels sont appliquées ces transformations géométriques. G peut contenir d'autres acteurs... qui eux-mêmes contiennent des acteurs, etc.; la structure est arborescente.

Tout acteur est la copie conforme de l'objet graphique

```
acteur:=begin_structure
            tran:=translate by p ;
            rota:=rotate o ;
            scal:=scale by t ;
            coul:=set color c, s ;
            cont:=set contrast to k ;
            appl:=instance of G ;
        end_structure ;
```

de plus, et c'est cela qui permet l'animation, les paramètres (p,o,t,(c,s,k)) peuvent être modifiés par les réseaux de fonctions cadencés par les données comme l'indiquent les figures 3.22 à 3.24 du chapitre 3

Les acteurs ne se différencient les uns des autres que par les valeurs affectées aux quintuplets (p, o, t, a, G).

A chaque acteur seront associées des suites de ces quintuplets, ce pendant la phase de mise en scène puis, après interpolation, pendant l'animation.

Certains acteurs peuvent ne jamais être modifiés, nous les appellerons des **décors**.

A un moment donné, tous les acteurs sont dans une certaine position et orientation et possèdent chacun leurs attributs de couleur et de luminosité; l'ensemble de tous les paramètres ainsi figés constitue ce que nous appelons une **scène**.

Mise en scène

Le metteur en scène peut modifier à l'aide des boutons et touches ces scènes et définir ainsi des **scènes-clés**.

A chaque scène-clé est associée une **date**. L'utilisateur peut fixer la date de son choix à l'aide d'une touche, définissant ainsi des repères de synchronisation avec des séquences réalisées précédemment.

La mise en scène consiste à établir une suite de scènes-clés.

Les opérations disponibles sont:

- **modification** des paramètres des acteurs de la scène visibles à l'écran
  - translation
  - orientation
  - changement d'échelle
  - modification de la couleur, du contraste.
- **datation** d'une scène
- **insertion** d'une scène-clé
- **suppression** d'une scène-clé

Ces opérations sont réalisées par l'intermédiaire de réseaux cadencés par les données (introduits au chapitre suivant) représentés figures 2.2, 3.21, 3.24.

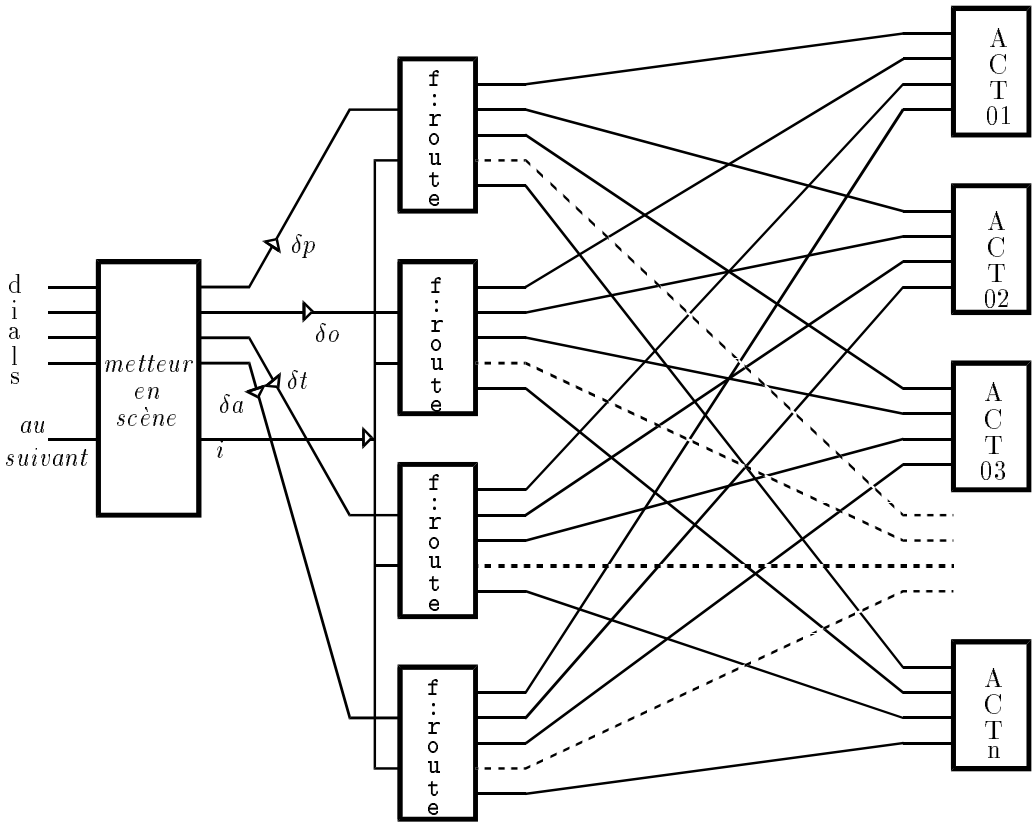


Figure 2.2: Le metteur en scène dirige les acteurs. Trois boutons sont associés à la translation p, trois autres à la rotation o, un au coefficient d'homothétie t, le dernier bouton disponible parmi les huit, peut être affecté, soit à la couleur c, soit à la saturation s, soit au contraste k. Ceci pour un seul acteur à un moment donné, le choix de l'acteur actif se fait à l'aide de la touche *au suivant*. Les valeurs relatives δp, δo, etc., issues des boutons sont aiguillées vers l'acteur actif numéro i par les boîtes de routage `f:route`.

Une séquence d'animation se déroulant à l'écran peut à tout moment être interrompue, les acteurs ainsi figés peuvent être modifiés définissant une nouvelle scène-clé à laquelle peut être associée une date, qui est généralement celle qui correspond à l'arrêt sur image.

Les notions de *réalisation* de la séquence d'animation et de *visionnage* (qui est ici appelée animation) sont de ce fait liées.

Caméraman

Lors de la mise en scène, l'utilisateur peut aussi manipuler les boutons des paramètres de la caméra [fig.2.3]. Celle-ci est définie par

- la position d'où l'on regarde
- l'endroit où l'on regarde
- le type de projection (parallèle ou perspective)
- le champ de vision
- la distance focale
- la profondeur de champ
- le type de vue: simple, gauche, droite ou stéréoscopique

Ces paramètres, traités de la même manière que ceux des acteurs sont également enregistrés puis interpolés.

Ainsi il est facile de réaliser des *travellings*, plongées et contre-plongées, changements de focale (du téléobjectif au grand angle).

L'opération qui consiste à faire tourner la caméra autour de la scène a été réalisée pour des raisons de performances par la simulation d'un plateau tournant.

La programmation de la vision stéréoscopique, qui peut être réalisée de multiples façons, a été faite de manière que les différentes vues soient compatibles avec les deux types de projections, parallèle et perspective, indépendemment des dimensions des acteurs, étant entendu que l'effet produit n'est pas le même suivant la position de l'observateur par rapport à la scène, un objet situé à 20 cm n'est pas vu de la même manière qu'un autre à 10 m, même si les dimensions relatives sont identiques.

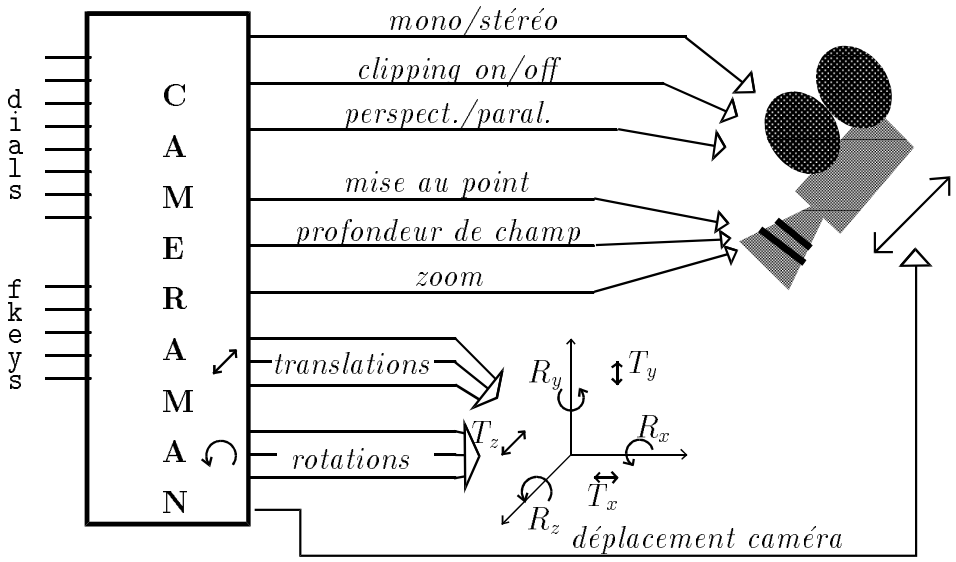


Figure 2.3: Le caméraman

Intervalliste et rôle des acteurs

La suite ordonnée, en fonction de la date, des scènes-clés enregitrées au cours de la mise en scène sur l'ordinateur hôte, constituent la trame de la séquence d'animation. Les paramètres relatifs à chacun des acteurs doivent être calculés par interpolation pour des intervalles de temps réguliers. Différents types d'interpolation ont été mis au point, il est possible entre autre, de donner plus ou moins de rondeur à la trajectoire, de rendre plus ou moins brusque les modifications de couleur etc.. Les interpolations sont faites en tenant compte des dates des scènes-clés, les vitesses de parcours des trajectoires sont calculées de manière à être le plus lisse possible.

La suite des valeurs ainsi obtenues constitue le **rôle** de l'acteur.

Animation

Les rôles sont renvoyés à la station graphique où ils sont affectés à chacun des acteurs à une cadence régulière, éventuellement modifiable par l'utilisateur devenu spectateur.

Les boutons du metteur en scène restent néanmoins actifs, l'utilisateur peut manipuler les acteurs qui jouent leur rôle. Un objet graphique n'est plus une simple construction 3D rigide mais une chose animée, vivante ...

A priori la cadence est fournie par une boîte unique `f:clockseconds`, il est néanmoins possible de commander les acteurs autrement, par des sources différentes ou les uns par rapport aux autres.

La scène qui se déroule sur l'écran est ce que voit la caméra virtuelle toujours disponible, toutes ses commandes étant actives.

Gestion des ressources

L'utilisateur agit sur les objets graphiques de l'écran en manipulant les boutons et touches de fonctions, chacun de ceux-ci pouvant avoir plusieurs significations. La gestion complète de toutes ces ressources physiques est faite par l'intermédiaire du réseau **réalisateur** comme le montre l'organigramme général figure 2.4.

2.4.3 Enregistrement par caméra ou magnétophone

La séquence d'animation créée sur le *PS300* doit pouvoir être diffusée sur un support *grand public* autre que le *PS300*.

Enregistrement vidéo

La solution a priori la plus simple consiste à en faire un enregistrement vidéo, malheureusement de sérieux problèmes sont apparus. L'enregistrement direct image par image ne peut être fait que si le *PS300* est équipé de l'option *sortie vidéo* fournie par le constructeur. Sans cette carte électronique il faut filmer l'écran à l'aide d'une caméra vidéo avec les inconvénients suivants

- l'enregistrement doit être fait à **vitesse réelle** (sauf si l'on dispose d'un équipement trés sophistiqué), ce qui peut ne pas être possible si la séquence d'animation est un tant soit peu étoffée, le *PS300* n'ayant plus le temps de rafraîchir l'image 60 fois par seconde.

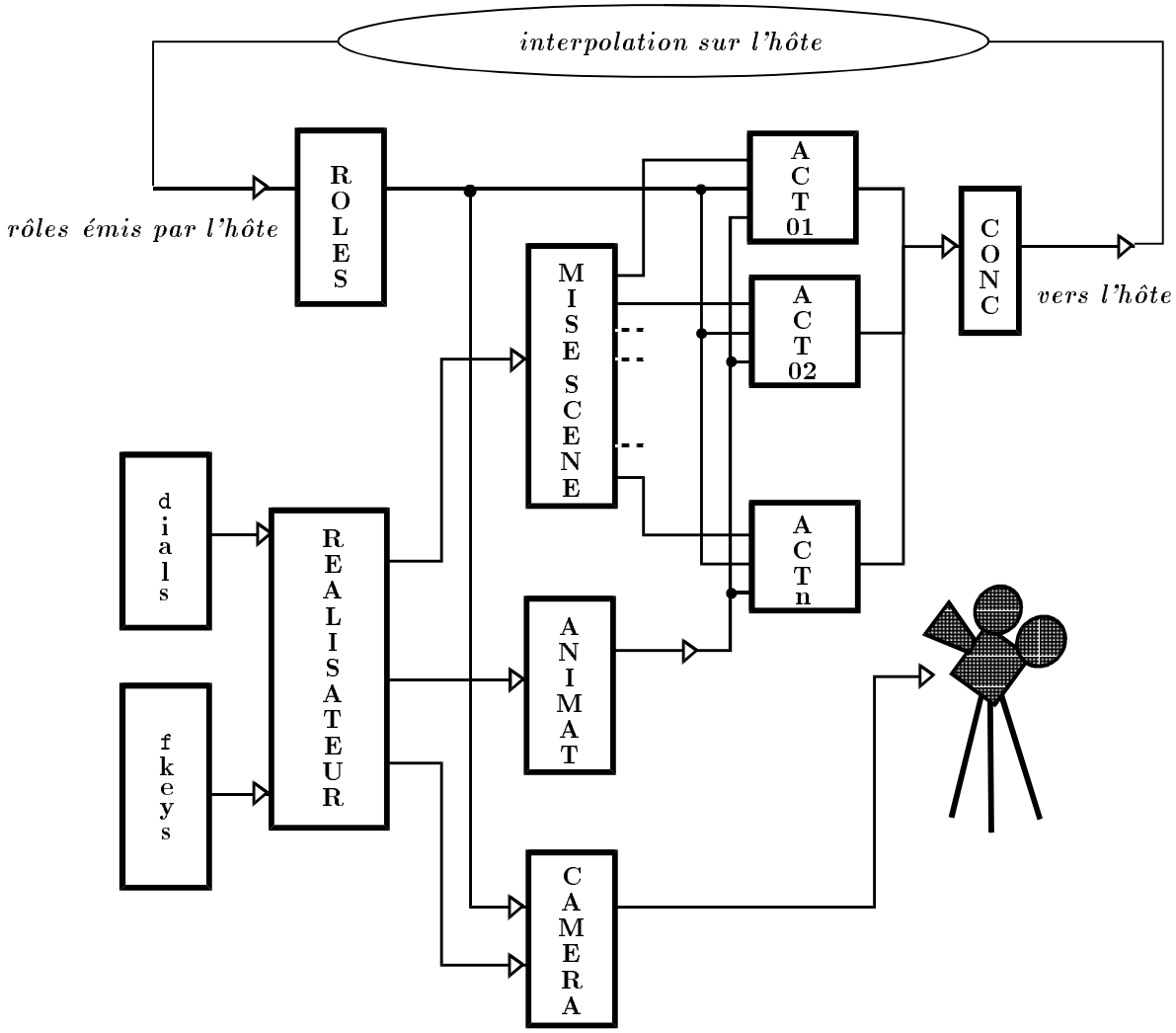


Figure 2.4: Organigramme général des réseaux de AGraph. Le *réalisateur* distribue les ressources *dials* et *fkeys*. Les *acteurs* sont dirigés interactivement par le *metteur en scène*. Les scènes-clés qu'il crée sont transmises à l'hôte sous la forme de chaînes de caractères (après concaténation). Les rôles obtenus par interpolation sont renvoyés aux acteurs qui peuvent être animés sur commande du réseau *animateur*. Une caméra virtuelle commandée par le *cameraman* permet la visualisation en temps réel sur l'écran.

- la qualité de la copie est relativement médiocre, l'image n'est pas toujours nette, le rendu des couleurs n'est pas bon, la sensibilité de la caméra n'étant pas identique sur tout le spectre.

L'enregistrement vidéo permet néanmoins de procéder à des essais. Mis à part l'investissement initial dû au matériel, les coût sont minimes et l'on dispose immédiatement des résultats. Suivant le type de matériel disponible on peut procéder soi-même au montage et éventuellement réaliser la sonorisation.

Prise de vue par caméra

Nous avons été amenés pour différentes raisons à filmer l'écran à l'aide de caméras cinéma 16 ou 35mm. Les détails techniques sont présentés en annexe.

L'enregistrement des séquences se déroulant à l'écran est souvent difficile et toujours fastidieux, il nous a paru nécessaire de développer des outils logiciels et matériels pour cela.

Afin de faciliter le montage de séquences réalisées à des moments différents, la commande de la caméra doit obligatoirement être faite par le *PS300* lui-même. Il faut pouvoir arrêter la séquence à tout moment, et ce de telle sorte que la dernière image ait été exposée correctement, que l'obturateur soit fermé et que la caméra soit prête à enregistrer l'image suivante.

La cadence d'enregistrement ainsi que le temps d'exposition peuvent être commandés par logiciel. En ce qui concerne le temps d'exposition, deux techniques ont été utilisées, l'une mécanique, qui consiste à fermer l'obturateur (tournage de *Boy*, voir chapitre 6), l'autre à n'afficher l'image que pendant le temps nécessaire à l'exposition (tournage de l'hologramme, voir chapitre 6). Le règlage du temps d'exposition s'est avéré être un problème délicat, les contrastes et les couleurs n'étant souvent pas corrects, il nous est même apparu que le temps d'exposition pouvait éventuellement être calculé en fonction de l'image affichée (il suffit de savoir combien de traits sont tracés). Les résultats ne sont malheureusement pas toujours satisfaisants, la qualité de l'image affichée par l'écran n'est de loin pas atteinte sur la pellicule.

Néanmoins, le tournage à l'aide de matériel professionnel a permis la réalisation d'hologrammes nécessitant une relative grande précision.

2.4.4 Ce qu'il faudrait encore faire

- Lors de la mise en scène, il serait utile de pouvoir introduire des positions clés pour une partie des paramètres de certains acteurs seulement, ceci afin de pouvoir affiner leur interpolation sans avoir à préciser les positions intermédiaires de tous.
- Il arrive quelquefois qu'un acteur doive évoluer d'une manière précise, calculable à l'aide d'une équation, il peut par exemple suivre une trajectoire parabolique en tournant sur lui-même régulièrement. A priori une interpolation de scènes-clés peut ne pas fournir le résulat souhaité, le rôle doit être calculé explicitement à l'aide de l'équation. On retrouve ici les raisons des développements des logiciels d'animation décrits en 2.3.2.
- Les mouvements d'un des acteurs peuvent être déduits de ceux d'un autre, par exemple la trajectoire et l'orientation de l'un doivent être les miroirs de celles de l'autre.
- etc.

Ces quelques développements faciliteraient beaucoup le travail du metteur en scène, leur mise au point ne présente aucune difficulté, par contre, leur utilisation peut être délicate et nuire à la simplicité de la mise en scène par interpolation de scènes-clés.

2.5 Les applications de *A*Graph

Au chapitre 6 sont décrites les principales applications d'animation graphiques interactives auxquelles ont donné lieu ce travail.

Nous tenons à signaler ici que *A*Graph s'est révélé être un logiciel à usage multiple, pouvant servir à autre chose qu'à la réalisation de séquences d'animation.

Le fait de pouvoir manipuler des acteurs animés donne une dimension supplémentaire à la notion de graphisme interactif, surtout s'il est utilisable facilement, sans programmation, donc accessible à tout utilisateur.

Les séquences d'animation peuvent également servir de vecteur d'information entre différents laboratoires. Le transfert (via réseau) des scènes-clés et de la définition

des acteurs étant suffisant si le destinataire dispose du logiciel *A*GRAPH. Celui-ci n'aura plus qu'à recréer les rôles par interpolation puis admirer la séquence à l'écran étant entendu qu'il dispose des *vector_lists* constituant les acteurs, par exemple les coordonnées tridimensionnelles des atomes de molécules biologiques ou la structure filaire d'un objet obtenu par CAO.

2.6 Conclusion

*A*GRAPH devait être un logiciel accessible à tous sans aucune programmation et permettre à tout utilisateur de manipuler des objets animés. Mis à part la création des objets à visualiser, qui doivent provenir d'autres logiciels, *A*GRAPH est utilisable directement et ne nécessite aucune connaissance du mode de fonctionnement de la station graphique.

Les séquences d'animation créées par *A*GRAPH sont d'une qualité acceptable mais ne doivent pas être considérées comme étant du cinéma. Le **cinéma** est un **art**, il est fait par des **professionnels** avec des moyens adaptés. L'animation graphique interactive telle que la permet *A*GRAPH n'est qu'une façon d'utiliser correctement une station graphique, à des fins de recherches ou d'outil pédagogique.

*A*GRAPH pourrait être implanté sur d'autre stations graphiques mais cela nécessiterait la réécriture d'une grande partie du logiciel, le *PS300* étant unique.

Chapitre 3

Cadencement par les données

3.1 Introduction

Les progrès en électronique, les gros besoins en puissance de calcul et en rapidité de temps de réponse, ainsi que les nouvelles techniques de programmation ont été la cause des évolutions dans l'architecture des ordinateurs. Leur principe de fonctionnement n'est plus seulement basée sur le modèle *von Neumann*, où un seul processeur exécute une à une les instructions d'un programme séquentiel sur des données stockées dans une mémoire centrale, mais de plus en plus sur des notions de décentralisation et de parallélisme. L'important n'est plus simplement l'**instruction** qui doit être exécutée mais l'**information** devant être produite. Ce sont les **données** qui demandent à être traitées, d'où la notion de **cadencement par les données**. L'exécution n'est plus contrôlée par un organe central unique, le **compteur ordinal**, mais peut être répartie sur plusieurs entités réelles ou virtuelles. Le *cadencement par les données* s'applique bien au parallélisme [Gajski *et al.*, 1982], à la mise en œuvre des systèmes réactifs, plus particulièrement aux applications graphiques et bien sûr à *A*Graph.

3.2 Parallélisme et programmation

Le développement des circuits intégrés [Computer Architecture 1985] permet maintenant la mise au point d'ordinateurs *MIMD (Multiple Instructions, Multiple*

Data) pouvant exécuter sur un trés grand nombre de processeurs des programmes différents *(MI)* traitant chacun ses propres données *(MD)*, répondant ainsi aux besoins en puissance de calcul des grosses applications. Le **parallélisme** effectif, rendu possible par de telles architectures [Duncan 1990], ne peut être pleinement exploité que si l'on dispose des outils logiciels adéquats: langages, compilateurs et systèmes d'exploitation [Dennis 1980]. En effet, les **limites** à l'utilisation des machines parallèles sont dues en grande partie à leur **programmation**; peut être parce qu'il est difficile de programmer *parallèle* mais surtout parce qu'il faut adopter un nouveau style de programmation. A ce propos, David Gelernter nous rappelle, dans [Gelernter 1986], qu'en *1957* le but des concepteurs de *Fortran* était l'*élimination de la programmation...*, ce but a été atteint, parce qu'à l'époque, *programmer* signifiait autre chose (planter des fiches ou aligner des lignes de code machine), depuis les niveaux de concept et de complexité ont changé.

3.2.1 Les raisons du parallélisme

Deux raisons essentielles ont conduit à envisager le parallélisme, la puissance de calcul, évidemment, mais aussi la possibilité d'utilisation de ce parallélisme dans l'expression du problème.

- L'augmentation régulière (et exponentielle) de la vitesse des circuits électroniques depuis plus de 40 ans a toujours permis de répondre aux besoins du moment. Mais, bientôt, les limites théoriques des technologies employées seront atteintes, et les futurs gains de puissance ne pourront être faits que grâce au développement des ordinateurs parallèles.
  - La **puissance de calcul** nécessitée par le calcul scientifique, les applications d'intelligence artificielle ou l'informatique graphique par exemple, ne pourra plus être fournie par des ordinateurs monoprocesseurs.
  - Par ailleurs, les **temps de réponse très courts** demandés par des applications temps-réel de surveillance de processus industriels ou de programmes interactifs, par exemple, ne peuvent souvent être respectés que lorsque plusieurs tâches sont exécutées simultanément par des processeurs différents.

- La recherche d'une solution séquentielle à un problème donné n'est pas nécessairement simple, au contraire. L'écriture d'un programme séquentiel n'est *naturelle* que s'il s'agit de coder le comportement, instruction par instruction, de la machine. Un concept mathématique, un graphe de flots de données ou un "programme" *Prolog* n'ont aucune raison d'être *linéarisés*. Le parallélisme peut amener une certaine facilité, ne serait-ce que dans l'expression du problème par le langage de programmation. *Can programming be liberated from the von Neumann style?* [Backus 1978].

3.2.2 Architecture "von Neumann", langages impératifs

La plupart des langages de programmation classiques sont destinés aux ordinateurs ayant une architecture de type **von Neumann** *(SISD)* où un seul processeur exécute une à une des instructions élémentaires. Celles-ci se résument aux opérations de lecture-écriture en mémoire, addition, soustraction, multiplication, division, comparaison, saut conditionnel, entrées-sorties et gestion des interruptions. L'exécution purement **séquentielle** est guidée par un **compteur ordinal unique** qui pointe vers une **mémoire centrale** contenant le programme et les données.

Ces langages, et les habitudes des programmeurs, ont fait que la programmation consiste à indiquer à l'ordinateur quelles opérations il doit effectuer pour trouver la solution du problème posé. On les appelle des langages *impératifs* ou procéduraux. Ils permettent d'énoncer l'algorithme et non pas de décrire le problème. Ils sont liés à la machine et surtout au mode d'exécution séquentiel; ils sont de ce fait mal adaptés pour exprimer simplement le parallélisme éventuellement inhérent à un problème donné.

3.2.3 Les langages de programmation du parallélisme

Ces langages, D.Gelernter les classe en trois catégories [Gelernter 1986]:

- les langages dérivés des langages *impératifs* ("Algol-based",d'après D.Gelernter),
- les langages issus de l'*Intelligence Artificielle (MultiLisp, Concurrent Prolog)*,

- les langages fonctionnels et les langages à flots de données [Agerwala et Arwind, 1982];

nous mentionnerons également, sans vouloir leur attribuer un classement particulier,

- les langages *orientés objet*,
- les langages de programmation des systèmes réactifs.

L'existence de ces nombreux langages, ayant des possibilités et des contraintes souvent très variées, est dues, pour une part aux domaines d'applications différents, mais traduit aussi la difficulté de trouver une solution à la programmation, qu'il y ait ou non parallélisme.

Langages impératifs et parallélisme

Certains compilateurs sont capables d'extraire, seuls, un peu du parallélisme contenu dans un programme écrit en un langage de programmation séquentiel "à la *von Neumann*" (par exemple compilateurs *Fortran* ou *C* pour processeurs vectoriels ou machines multiprocesseurs). C'est une solution intéressante, qui permet de paralléliser tous les programmes existants, et il y en a beaucoup, sans les réécrire..., mais ce n'est pas toujours possible et rarement optimal: l'énoncé du problème sous la forme d'un programme en langage impératif ayant déjà fait perdre trop d'informations et ajouté des contraintes inutiles. Rien ne permet de dire à l'heure actuelle que des programmes mêmes très *intelligents* résoudront un jour ce problème: le programmeur doit fournir à ces compilateurs des indications concernant le parallélisme. Cela peut sembler être une charge supplémentaire, mais, comme le souligne C.Seitz, cité dans [Gelernter 1986], programmer un ordinateur parallèle, le Cosmic Cube par exemple, n'est finalement pas plus difficile que de programmer en un langage séquentiel classique, à condition de disposer des outils adéquats.

Ainsi, programmer explicitement une application exécutant en parallèle plusieurs processus revient, d'une part, à décrire chaque processus et, d'autre part, à indiquer où et comment ils doivent échanger des informations et se synchroniser [Andrews et Schneider, 1983]. Ces échanges se font par l'intermédiaire d'une mémoire commune ou par des canaux de

communication. Ces langages sont, soit spécialement créés pour la programmation du parallélisme (*CSP* [Hoare 1978], *Occam*), offrant des outils qui permettent l'expression du parallélisme à grande échelle, soit des adaptations de langages existants (compilateurs pour machines vectorielles ou multiprocesseurs, *Concurrent Pascal, Concurrent Fortran, Concurrent C, Ada*, etc.).

La répartition des tâches sur les différents processeurs se fait en tenant compte des flots de données qui sont sous-jacents à ces échanges, qu'ils soient spécifiés explicitement par le programmeur ou détectés à partir des graphes de dépendances des données par le compilateur parallélisant [Allan 1980].

Les langages de l'IA

Après les extensions parallèles de *Lisp* ou *Prolog*, les développements dans ce domaine ont souvent mené à des systèmes fondés sur des architectures spécialisées (machines *Lisp*, ou projets japonais d'ordinateurs de cinquième génération [Treleaven et Lima, 1982]).

Langages fonctionnels

Par opposition aux langages *impératifs* qui obligent le programmeur à donner explicitement une solution, les langages *fonctionnels* l'amènent à énoncer le problème, en le décomposant en une suite d'expressions fonctionnelles. Il doit définir *"quoi calculer"*, plutôt que *"comment calculer"* [Hudak 1989].

Parmi les langages fonctionnels, citons, entre autres (voir [Hudak 1989]), le *lambda calcul* de A. Church [Church 1941], *Lisp* de J. McCarthy [McCarty 1963], *FP* de J. Backus [Backus 1978], *Lucid* de W. Wadge et E. Ashcroft [Wadge et Ashcroft].

Les langages fonctionnels n'utilisent pas de variables (au sens habituel du terme), n'ont donc aucun effet de bord et surtout ne font pas référence à la machine. Il est ainsi facile d'en extraire le graphe de dépendance des données et de concevoir une implantation parallèle *efficace* [Ackerman 1982], [Berger *et al.*, 1987].

Langages orientés objet

Les notions d'*encapsulation* et d'*échanges de messages* introduites par les langages de programmation *orientés objet*, obligeant le programmeur à décomposer son application en modules relativement indépendants, facilitent l'implantation parallèle des différents modules.

On peut, là aussi, remarquer que l'élaboration d'un modèle conduisant aux *objets* se fait souvent, d'après G. Booch [Booch 1986], à partir d'un diagramme des flots de données.

Programmation des systèmes réactifs

Les automaticiens se sont intéressés depuis longtemps à la programmation des automates de contrôle de processus industriels ou de systèmes embarqués et ont développé de nombreux langages (en dehors des langages de programmation du parallélisme classiques *CSP, Ada et Occam...*) parmi lesquels nous citerons le *Grafcet* (application industrielle des réseaux de Pétri)[Petri 1975, Thelliez et Toulotte, 1982] ou les langages tels que *Esterel, Lustre* ou *Signal* ([Berry *et al.*, 1987]).

Leurs domaines d'application sont assez spécifiques, souvent il s'agit de gérer en parallèle un grand nombre de périphériques, et cela avec des contraintes de temps de réponse.

J'ai moi-même participé à l'automatisation de l'atelier chocolat de l'usine *Mars Alimentaire* de Hagenau, où j'ai été chargé de programmer en *Grafcet* un ordinateur *Solar 1640* gérant 700 entrées-sorties de différentes natures, tout-ou-rien, niveaux de cuves, températures, commandes de vannes, démarrage de moteurs, etc.

La programmation d'une telle application en *Grafcet* consiste à décrire chaque partie du système en termes d'états d'automate dont les transitions d'un état à un autre sont déclenchées par les périphériques et les changements d'états des autres parties. Le graphe que l'on peut associer à un tel automate ne traduit pas directement les flots de données. Celui-ci est par contre à la base de la programmation des transitions.

Flots de données et parallélisme

L'analyse par un compilateur d'un des langages précédents conduit ainsi généralement à une interprétation en tant que graphe de flots de données. Nous abordons maintenant cette question sous l'aspect des *langages de programmation à flots de données.*

Citons *VAL (Value oriented Algorithmic Language)* [Ackerman et Dennis, 1979], *Id* et *Id Nouveau* [Arwind *et al.*, 1978], *Lucid* [Wadge et Ashcroft], *LAU (Langage à Assignation Unique)* [Berger *et al.*, 1987], et les réseaux cadencés par les données (*DDN (Data Driven Network)* [Davis 1978]), qui nous intéresseront particulièrement.

3.3 Réseaux cadencés par les données

Nous présentons ici les réseaux de fonctions cadencés par les données, introduits par A.L. Davis dans [Davis, Davis 1978], qui sont utilisés comme base du langage de programmation de la station graphique *PS300* Evans & Sutherland.

Ce langage, que nous appelons *"PSDDNL"*, décrit en annexe A, a été largement utilisé pour la réalisation du logiciel *A*GRAPH.

3.3.1 Principe général

Contrairement à l'approche *von Neumann*, où *un* programme modifie séquentiellement, instruction après instruction, les données contenues dans *une* mémoire, ce sont les données elles-mêmes qui, dans un ordinateur cadencé par les données, circulent d'une fonction à une autre. Ces fonctions sont activées lorsque toutes les données nécessaires à leur exécution sont présentes. Leurs résultats sont envoyés sur les entrées d'autres fonctions.

Graphe associé à un réseau

Un réseau cadencé par les données peut être considéré comme un graphe orienté dont les sommets représentent des fonctions et dont les arcs servent à convoyer les valeurs produites et consommées par celles-ci. Chaque arc incident à un nœud

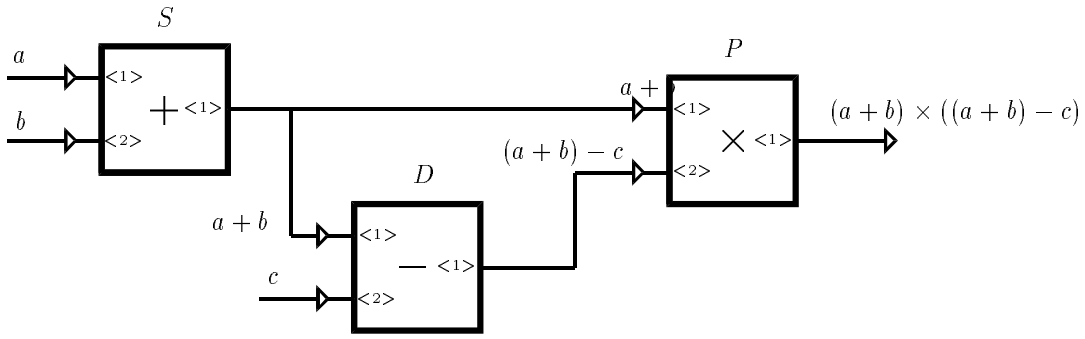


Figure 3.1: Ce réseau, composé des trois fonctions $+$, $-$ et $\times$, fournit un résultat $(a+b) \times ((a+b)-c)$, fonction des entrées a, b et c.

correspond à une des entrées de la fonction symbolisée par ce nœud, chaque arc sortant correspond à un des résultats.

Activation des fonctions, files d'attentes

Sur la figure 3.1 est représenté le réseau construit à l'aide des trois fonctions S, D et P, ayant chacune deux entrées et une sortie, définies par

$$S(x,y) = x + y, \qquad D(x) = x - y, \qquad P(x,y) = x \times y.$$

La sortie de S est connectée à la fois sur l'entrée `<1>` de D et sur l'entrée `<1>` de P. La sortie `<1>` de P s'écrit, en fonction de a, b, c,

$$P(S(a,b), D(S(a,b),c)) = (a+b) \times ((a+b)-c)$$

L'exécution de ce réseau se déroule de la manière suivante: lorsque toutes les données d'une fonction sont disponibles celle-ci peut être évaluée. Par exemple, S est activée par l'arrivée de a et b sur ses deux entrées. La valeur $a+b$ est calculée et envoyée à la fois sur l'entrée `<1>` de D et sur l'entrée `<1>` de R. Là elle sont mises en attente jusqu'à l'arrivée de c sur `<2>`D, d'une part, puis de (a+b)-c sur l'entrée `<2>`P, d'autre part.

Les entrées des fonctions sont gérées comme des files d'attente infinies: si plusieurs valeurs sont envoyées sur l'entrée `<1>`S, elles sont mémorisées, dans l'ordre,

puis traitées au fur et à mesure de l'arrivée des valeurs correspondantes sur l'entrée <2>.

Remarquons que les systèmes *Data Flow* ne fonctionnent pas tous d'après ce schéma, nous y reviendrons au paragraphe 3.5.

Temps d'exécution d'une fonction

Le temps d'exécution d'une fonction est supposé fini mais inconnu et non reproductible. Aucune hypothèse n'est faite sur l'ordre d'exécution des différents nœuds, sur l'ordre d'émission des résultats lorsqu'une fonction a plusieurs sorties, ni sur la vitesse de transmission des données sur les arcs.

Types des données, nature des fonctions

Les valeurs véhiculées par les arcs peuvent être des nombres, des chaînes de caractères, des matrices; les fonctions sont quelconques, par exemple les fonctions usuelles telles que somme de deux nombres, produit de matrices, extractions de sous-chaînes, comparaisons, opérateurs de conversion, mais aussi aiguillages, sélecteurs et multiplexeurs que nous étudierons en détails au paragraphe 3.3.3.

Entrées conjonctives, disjonctives et constantes

Les n entrées d'une fonction sont dites *conjonctives* si la présence d'une donnée sur chacune d'elles est nécessaire au calcul de la fonction.

Les entrées sont dites *disjonctives* lorsque la fonction peut être calculée sans que toutes les entrées soient nécessaires. Cette distinction entre fonction conjonctive et disjonctive est à considérer avec beaucoup de précaution. En effet, une fonction pourrait être disjonctive si elle existait en temps que fonction élémentaire sans que sa réalisation par un réseau le soit. Ainsi, par exemple, la fonction $P(a,b) = a \times b$ pourrait être calculée pour une valeur de $a = 0$ avant (et même sans) l'arrivée de b; d'autre part, si l'on considère la fonction composée $H(a,b,c) = P(S(a,b), D(S(a,b),c)) = (a+b) \times ((a+b) - c)$, qui est à entrées conjonctives, et qui est réalisée par le réseau figure 3.1, son exécution *peut être commencée* avant l'arrivée de c.

Pour des raisons de simplicité de représentation de réseaux, nous introduirons des fonctions ayant une ou plusieurs entrées *constantes*, c'est-à-dire des entrées qui sont initialisées lors de la création de la fonction, qui sont utilisables indéfiniment et cela sans être consommées (ce type d'entrées existe pour les fonction du langage "PSDDNL"). Sauf cas particulier, ces entrées constantes ne doivent pas être modifiées en cours d'exécution car le réseau ne serait plus déterministe.

Notations et conventions graphiques

Les fonctions sont aussi appelées *boîtes*, les arcs qui les relient sont appelés *fils*.

Afin de ne pas surcharger les dessins des réseaux, nous adopterons, sauf mention explicite du contraire, les conventions suivantes:

- les entrées sont situées à gauche des boîtes,
- les sorties à droite,
- il n'y a pas de connection entre deux fils qui se *croisent*,
- il y a toujours connection dans un branchement en `T` (à trois brins),
- pour les branchements à plus de trois brins, la connection est indiquée par un •,
- les flèches indiquant le sens de circulation des données ne sont là que pour faciliter la lecture du réseau, elles ne sont presque jamais nécessaires (sens = sortie (à droite d'une boîte) vers entrée (à gauche)),
- les entrées de boîtes non raccordées sont en principe des initialisations faites lors de la création de la boîte, c'est le cas de l'initialisation à *vrai* du selecteur de la figure 3.7, ou de l'initialisation à 1 des boîtes notées $1, P, P, \cdots, P$ et $1, x, x, \cdots, x$ de la figure 3.11,
- les données émises sur des sorties non connectées ou connectées à la masse sont perdues.

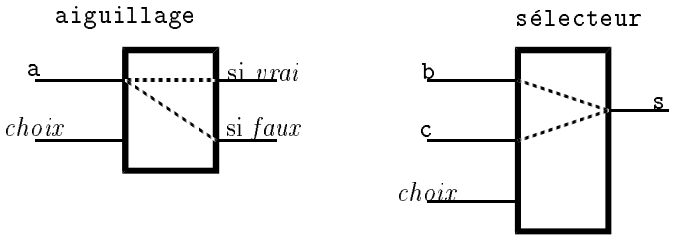


Figure 3.2: Fonctions de routage: La valeur `a` est émise sur l'une des deux sorties en fonction de la valeur logique *choix* de l'**aiguillage**. L'une des deux valeurs (`b` si *vrai* ou `c` si *faux*) est émise sur la sortie `s` en fonction de la valeur de l'entrée *choix* du **sélecteur**.

3.3.2 Fonctions de base

Le programmation en **langage machine** d'un processeur suppose l'existence de fonctions cablées ou microprogrammées qui constituent les instructions élémentaires de base. De la même manière, un langage de programmation cadencé par les données est construit sur un ensemble de fonctions élémentaires dont l'exécution n'est à priori pas décomposable.

Le choix de cet ensemble de boîtes élémentaires est spécifique à chaque langage, et dépend de la ou des machines sur lesquelles il est implanté ainsi que des domaines d'application. En annexe figure la liste des fonctions élémentaires du langage du *PS300* Evans & Sutherland. Ces réseaux de fonctions sont simulés par un processeur MC68000 classique et sont plus orientés vers les possibilités de manipulation d'une stucture de données graphique.

Comme pour les circuits logiques, un ensemble réduit à la fonction `NAND` ou aux trois fonctions `AND`, `OR` et `NOT`, manipulant des données logiques *vrai* et *faux*, serait suffisant pour *tout* calculer. A l'autre extrême, un ordinateur quelconque peut être considéré comme réalisant une *fonction élémentaire* sur les données qu'il reçoit des autres machines du réseau d'ordinateurs.

3.3.3 Fonctions de routage

La réalisation d'un réseau cadencé par les données nécessite, dans la plupart des cas, l'utilisation de fonctions de routage.

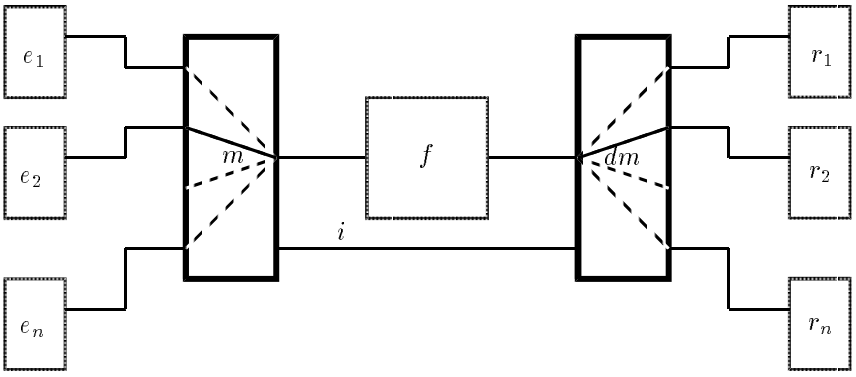


Figure 3.3: Le multiplexeur m et le démultiplexeur dm permettent de transmettre la donnée issue d'une fonction émettrice e_i à un réseau f, puis d'aiguiller le résultat vers la fonction réceptrice correspondante r_i. Le numéro i de la donnée courante est transmis de m vers dm.

- l'une, appelée *aiguillage*, permet l'émission, sur l'une ou l'autre des deux sorties, d'une entrée en fonction de la valeur *vrai* ou *faux* de l'autre entrée. C'est une fonction *élémentaire*, non décomposable; nous la représentons graphiquement comme sur la figure 3.2

- l'autre, appelée *sélecteur* permet de choisir une parmi deux entrées en fonction de la valeur *vrai* ou *faux* d'une troisième, elle est symbolisée par la boîte de la figure 3.2. Le sélecteur est disjonctif, c'est-à-dire que l'entrée *non choisie* peut ne *pas être présente*. Si elle est présente, elle n'est pas consommée, mais sera prise en compte pour la prochaine valeur ad hoc de l'entrée *choix*.

- d'autres boîtes de routage peuvent être construites, par exemple les fonctions *multiplexeur-démultiplexeur* (fig. 3.3) qui permettent de gérer simplement le partage d'une ressource comme le montre la figure 3.3 où les accès à la fonction f par les différentes valeurs issues des boîtes e_i sont canalisées par le multiplexeur m, la valeur correspondante calculée par f est envoyée au bon destinataire par dm.

- la fonction de *synchronisation* (à 2, 3 ou n entrées) émettant sur chaque sortie l'entrée correspondante lorsque toutes celles-ci sont arrivées, permet de construire des réseaux qui peuvent ne pas être déterministes (mais qui le sont en

pratique si on *laisse le temps* à la machine d'exécuter ses fonctions); nous en présentons une application au paragraphe 3.3.4.

Si la réalisation de circuits *séquentiels*, c'est-à-dire dont le graphe ne comporte pas de boucles, construits par composition de fonctions élémentaires est facile sinon évidente, il est par contre difficile de mettre en œuvre des réseaux comportant des *itérations* et des activations *conditionnelles* de portions de réseaux en fonction des valeurs circulant sur les arcs. Les choses se compliquent sérieusement lorsqu'il s'agit de créer des réseaux qui doivent fonctionner de manière totalement asynchrone, en permettant au maximum l'**exécution pipeline** de flots de données successifs, par un nombre quelconque de processeurs. C'est ce que nous illustrons maintenant par une série d'exemples concrets.

3.3.4 Un premier exemple, x^n

La réalisation d'un réseau itératif calculant x^n paraît à première vue facile, en effet, comme le montre la figure 3.4: "... il suffit de reboucler n fois sur la boîte $\times$, puis d'émettre le résultat...".

Mais..., certaines entrées ont plusieurs prédécesseurs (d'où non déterminisme et erreurs possibles), et le réseau n'est pas à proprement parler *pipeline* puisque le calcul de y^m ne peut commencer qu'après terminaison du calcul de x^n (c'est la boîte de synchronisation qui l'interdit).

Nous nous proposons de réaliser un réseau qui puisse calculer *à la volée*, en *pipeline* les valeurs x^n, y^m etc. au fur et à mesure de l'arrivée des données sur les entrées et ce dès que les fonctions sont disponibles. La difficulté d'écriture d'un programme *pipeline* correct est clairement mise en évidence, sur le même exemple, par W.Wadge et E.Ashcroft au cours du chapitre 4 dans [Wadge et Ashcroft].

Voici notre solution:

Création de suites de n éléments

Aux entrées `<1>` et `<2>` de la boite $\times$ doivent parvenir les suites $\{1, x, x^2, \cdots, x^{n-1}\}$ d'une part, et $\{1, x, x, \cdots, x\}$ d'autre part. Nous devons pouvoir créer des listes de n éléments et aussi disposer d'une fonction *adjonction en tête* qui permette d'émettre

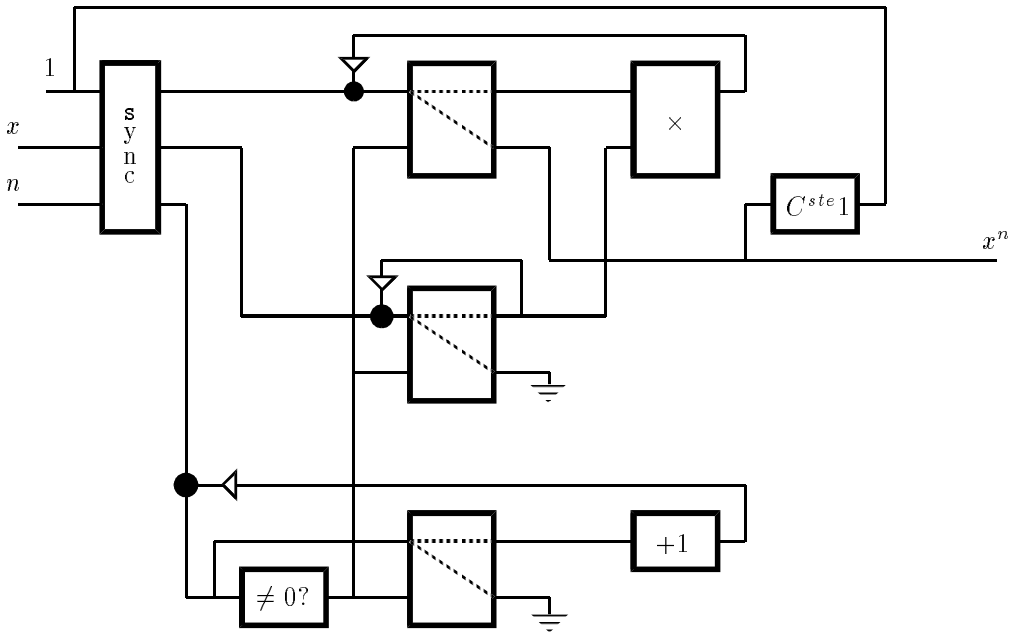


Figure 3.4: Ce réseau calcule x^n par itération. Mais il n'est pas optimal parce qu'il ne permet pas l'exécution *pipeline*, bloquée par la boîte de synchronisation, il n'est pas déterministe parce que plusieurs fils sont connectés à une même entrée (connections •)

.

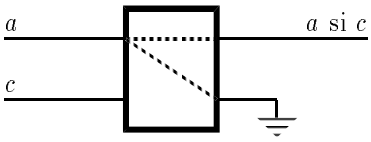


Figure 3.5: Le filtre $asic(a, c)$. L'entrée a est transmise si c est *vrai*, sinon elle est consommée.

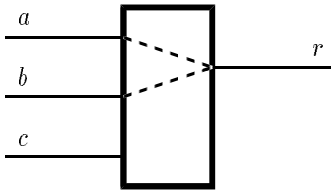


Figure 3.6: Réalisation de $sas(a, b, c)$ `= si` c `alors` a `sinon` b. L'entrée sélectionnée doit être présente, l'autre peut ne pas être là. Aucune entrée n'est perdue.

une constante (ici 1) avant une telle liste,... et qui réagisse de la même manière pour les prochaînes données arrivant en *pipeline* (c'est-à-dire que le sélecteur d'entrée est positionné correctement après émission du dernier élément de la liste).

Pour cela nous définissons les fonctions *filtres* suivantes:

- *asic(a,c)* qui réémet ou consomme l'entrée a selon la valeur *vrai* ou *faux* de la condition c (fig. 3.5); remarquons que les deux entrées doivent être présentes et sont consommées,
- *sas(a,b,c)*, qui correspond au *si-alors-sinon*, avec la particularité d'être à entrées disjonctives, c'est-à-dire que seule l'entrée choisie par la condition c doit être présente (fig. 3.6),
- *nà0(n)* (ou *nà1(n)*) qui émet les valeurs $n, n-1, \cdots, 0$ (ou $n, n-1, \cdots, 1$ respectivement) en fonction de la valeur $n \geq 0$ (ou ≥ 1) de l'entrée (fig. 3.7),
- *xnfois(x, n)* qui émet n fois l'entrée x, en consommant à chaque fois l'entrée x,
- *a_xnfois(a, x, n)* qui émet a puis n fois l'entrée x,

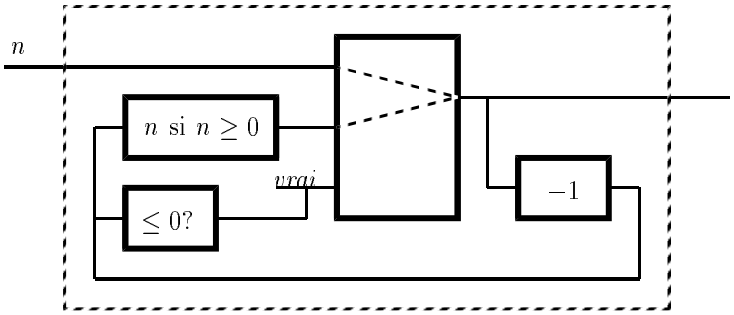


Figure 3.7: *nà0*(n) émet n, $n-1$, ..., 0. La $3^e me$ entrée du sélecteur est initialisée à *vrai* lors de la création de la boîte (voir 3.3.1).

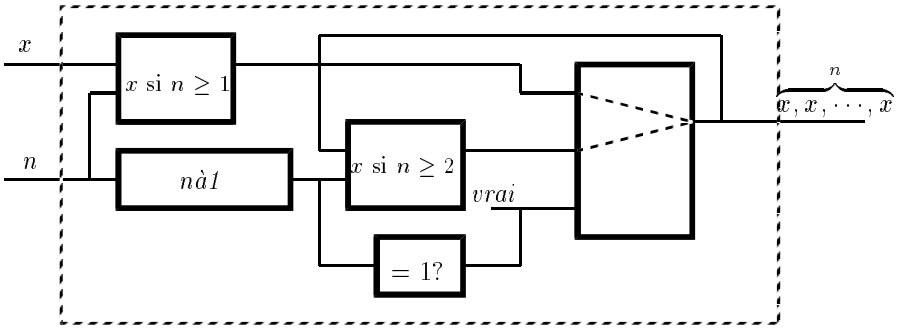


Figure 3.8: *xnfois*(x, x). L'entrée x est émise n fois.

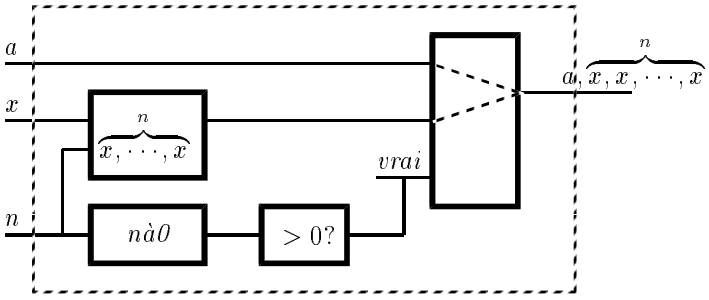


Figure 3.9: *a_xnfois*(a, x, n). Émission de a puis de l'entrée x, n fois.

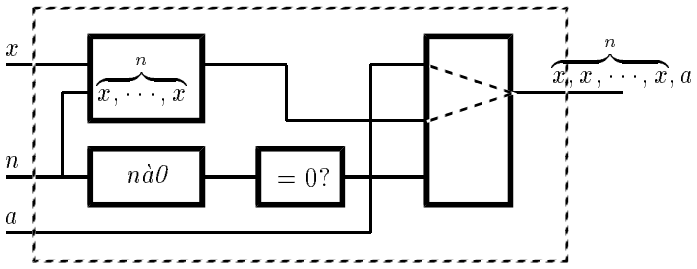


Figure 3.10: *xnfois_a*(x, n, a). Émission de l'entrée x, n fois, puis de a.

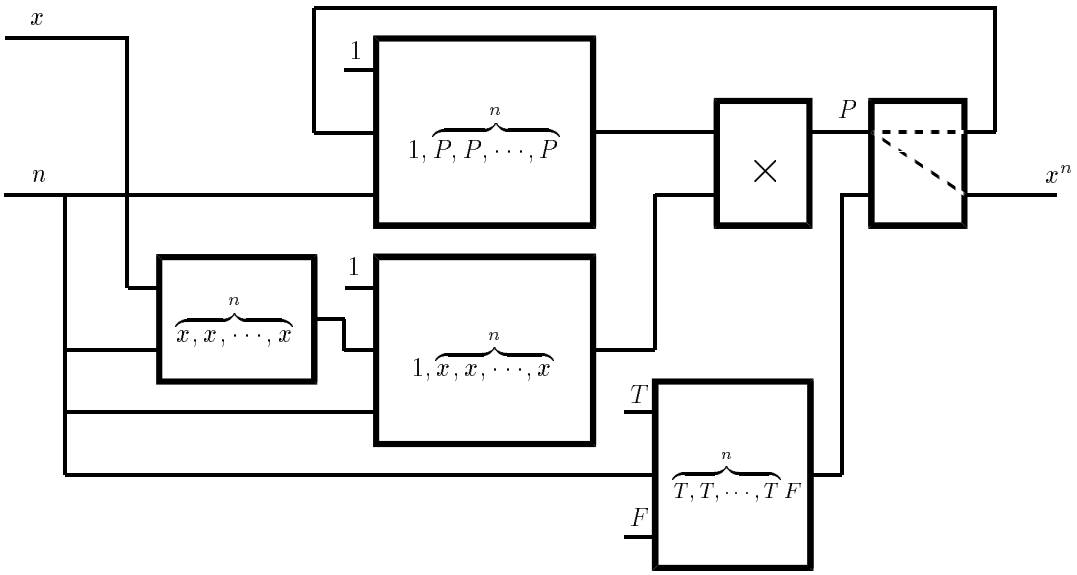


Figure 3.11: Ce réseau calcule x^n par itération.

- *xnfois_a(x, n, a)* qui émet n fois l'entrée x, puis l'entrée a.

Remarquons que la manipulation d'une suite finie d'éléments peut être envisagée de deux manières, soit la suite comporte une valeur particulière indiquant la fin de la liste, soit elle est accompagnée d'une information donnant le nombre d'éléments; c'est cette solution qui a été choisie ici.

Un réseau *pipeline*

Il est maintenant possible d'assembler ces fonctions pour créer le réseau de la figure 3.11 où toutes les boîtes peuvent s'exécuter de manière totalement asynchrone, même à l'intérieur des boîtes *xnfois* ou *a_xnfois*.

3.3.5 Un autre exemple, la manipulation de listes

Jusqu'ici, les informations circulant sur les fils entre les différentes boîtes étaient considérées comme des entités élémentaires. Une suite d'éléments émis par un portion de réseau peut être vue comme un objet unique, au sens d'une *liste* par exemple, et doit pouvoir être véhiculée, en tant que telle, d'un réseau vers un autre.

Sans entrer dans les détails, nous donnons quelques indications sur les possibilités de construction des fonctions qui sont nécessaires.

Transport d'une liste

Une liste ne peut pas être transmise par la simple émission de chacun de ses éléments, la boîte réceptrice ne pouvant jamais savoir quand arrivera le dernier élément. Plusieurs solutions peuvent être envisagées:

- émission du nombre d'éléments de la liste puis des éléments eux-mêmes,
- émission d'un indicateur de fin de liste (cela n'est pas toujours possible, en particulier si cet indicateur peut être pris comme valeur d'un élément),
- existence d'un fil de liaison parallèle indiquant le rang de la donnée transmise,
- etc.

Le problème se complique évidemment s'il s'agit de transmettre des listes de listes, des listes de listes de listes,...

Nous **admettrons dans ce qui suit** que les protocoles nécessaires à ces transports existent.

Fonctions de base

Nous supposons également que les fonctions de base suivantes sont disponibles:

- *crée(trig)*, qui crée une liste vide après réception d'une entrée *trig*,
- *vide(f)* émet *vrai* si la liste est vide,
- (t, c)=*têtecorps(f)*, qui permet de décomposer une liste f non vide en émettant l'élément situé en tête sur une des sorties, et le reste de la liste sur l'autre,

- *adjt*(x, f) réalise l'adjonction de x en tête de la liste f,
- *adjq*(f, q) l'adjonction en queue,
- *conc*(f_1, f_2), qui concatène f_1 et f_2.

Simulation d'une pile par inversion d'une file

La plupart des langages de programmation offrent la possibilité d'utilisation d'une pile *dernier entré-premier sorti*. Cette structure de données n'est certainement pas adaptée au cadencement par les données puisqu'il est impossible de la réaliser autrement que par inversion d'une liste. L'expression fonctionnelle équivalente (récursive) peut s'écrire:

inversion(d,f)=
si vide(f) **alors** d
sinon inversion*(adjt(tete(f),d),corps(f))*

Le premier appel se faisant par $inversion(\phi, f)$. Sur le réseau de la figure 3.12 on voit que le nombre d'opérations nécessaires est de l'ordre de n^2 si la liste comporte n éléments (en comptant comme opération élémentaire l'exécution d'une fonction).

3.3.6 Notation fonctionnelle d'un réseau itératif

S'il est relativement facile de comprendre un réseau schématisé par un dessin, il est illusoire de vouloir en donner une définition écrite sans introduire une nouvelle notation capable de traduire ces notions de suites (infinies ou non) et d'exécution *pipeline*. Le langage de programmation *Lucid (The Data Flow Langage)* est à ce titre parfaitement adapté; les concepts et les techniques qu'il introduit sont difficiles, nous ne les aborderons pas ici et invitons le lecteur à se reporter à [Wadge et Ashcroft]. Nous constatons aussi que l'expression d'un problème sous une forme récursive n'est pas directement transcriptible en réseau cadencé par les données, l'*empilement* des entrées dû aux appels récursifs n'étant réalisé nulle part si le nombre de boîtes réellement créées est inférieur au nombre d'appels. Certains travaux concernant le cadencement par les données font ainsi mention de réseaux récursifs, qu'il est évidemment facile de dessiner, mais qui ne sont exécutables que dans certains cas très particulier

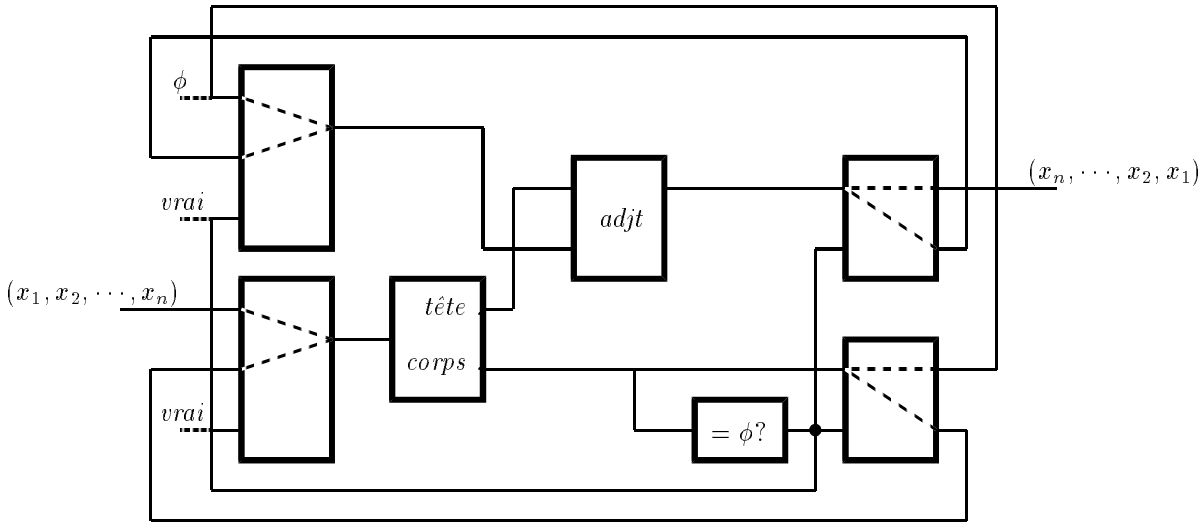


Figure 3.12: Ce réseau *inverse* une file non vide, le premier élément est émis en dernier, le deuxième en avant-dernier, etc. Il faut *relire* la file autant de fois qu'elle contient d'éléments.

(récursivité terminale par exemple) ou s'il existe un mécanisme "caché" permettant la récurrence.

3.4 Structures de données

Nous présentons maintenant quelques exemples de réseaux réalisant des structures de données plus compliquées. Pour des raisons de clarté de figures, nous nous permettrons certaines connections multiples étant entendu que celles-ci devraient être réalisées par des sélecteurs associés à des multiplexeurs.

3.4.1 La structure de données *ensemble*

Considérons par exemple un ensemble, au sens mathématique du terme, pour lequel sont définies les opérations de base suivantes:

inclusion d'un élément

suppression d'un élément

existence d'un élément

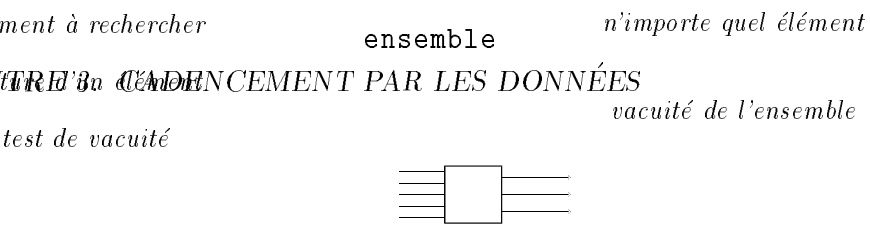


Figure 3.13: La *superboîte* `ensemble`.

test de vacuité de l'ensemble

obtention d'un élément quelconque

ainsi que les opérations plus complexes

pour tout élément de l'ensemble *répéter* ...

pour tout sous-ensemble *répéter*...

réunion, intersection de deux ensembles

etc.

Nous schématisons cette *structure de données* par la *superboîte* `ensemble` représentée figure 3.13. L'accès par plusieurs réseaux à cette *ressource* peut être géré par l'intermédiaire d'un multiplexeur-démultiplexeur défini figure 3.3.

L'ordre dans lequel arrivent les commandes a bien sûr une importance, mais ne dépend que de ces réseaux, c'est-à-dire que l'un peut, par exemple, inclure un élément x, dont un autre pourra plus tard tester l'existence. C'est aux réseaux eux-mêmes de se synchroniser.

Nous proposons trois implantations, la première, par liste circulaire, correspondant à un réseau dont la taille est fixe et indépendante du nombre de données traitées, la deuxième, entièrement parallèle, dont la dimension évolue en cours d'exécution au fur et à mesure que des informations apparaissent, la troisième correspondant un arbre binaire de recherche. Dans ces deux derniers cas, nous autorisons *l'allocation dynamique de ressources*, comme par exemple l'utilisation de nouveaux processeurs en cours de traitement, sans nous occuper pour le moment de la manière dont ceci peut être réalisé pratiquement (remarquons que sur la station graphique *PS300* cette allocation est faite par la fonction `f:h_chop` qui crée les programmes nécessaires à la simulation des boîtes).

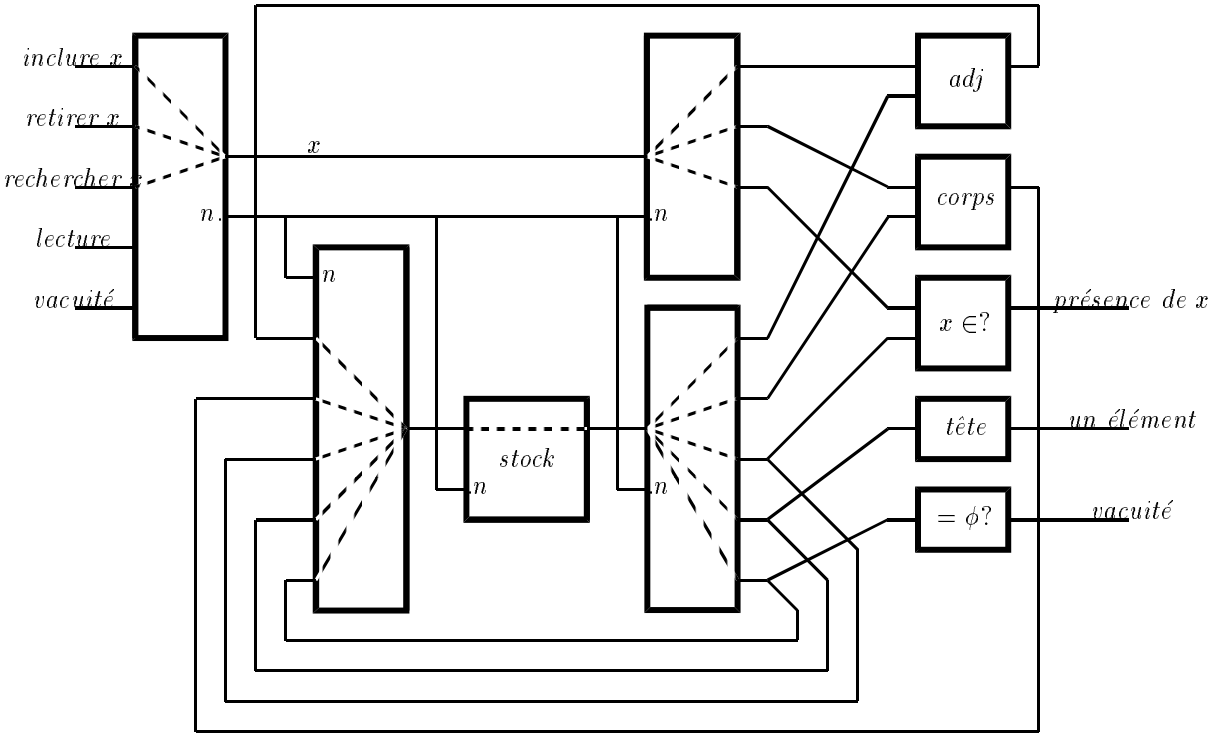


Figure 3.14: Implantation d'un ensemble comme liste de ses éléments. L'arrivée d'une valeur x sur l'une des entrées *insertion*, *suppression* ou *existence*, ou d'une demande *test de vacuité* ou *élément*, déclenche l'émission par la boîte *stock* de la liste des valeurs de l'ensemble. Cette liste est aiguillée en fonction de l'opération choisie vers l'une des fonctions, *adjonction*, *corps*, *appartenance*, *test de vacuité* ou *tête*. La nouvelle liste, ou l'ancienne s'il n'y a pas modification, est renvoyée, via un collecteur, vers la boîte de stockage qui n'est qu'une simple boîte de synchronisation mémoisant sur sa file d'attente d'entrée les éléments de la file (cette boîte n'est d'ailleurs pas nécessaire).

Implantation par liste circulaire

La figure 3.14 donne une implantation des fonctions de base à l'aide d'une structure de liste circulaire sur laquelle ont été greffées les opérations d'inclusion et de suppression d'un élément. La recherche d'un élément nécessite en moyenne $n/2$ opérations, l'inclusion est immédiate si l'on accepte la présence d'un même élément plusieurs fois dans la liste... mais complique sa suppression.

Implantation totalement parallèle

Les figures 3.15 et 3.16 correspondent à une implantation parallèle, l'adjonction d'un nouvel élément provoque la création du réseau figure 3.16, le test d'appartenance peut être rapide, **s'il existe**..., dans le cas contraire il est nécessaire d'attendre que l'arborescence collecte la réponse *faux*, ce qui ne peut se faire, au mieux, qu'en un temps $O(\log n)$, puisqu'il n'est pas possible de savoir combien de temps il faut attendre une éventuelle réponse *vrai*. Nous remarquerons que la recherche d'un élément active *toutes les boîtes*.

Implantation par arbre binaire trié

Les figures 3.17, 3.18, 3.19 et 3.20 présentent une implantation par arbre binaire trié d'un ensemble d'éléments munis d'une relation d'ordre totale. Le principe est le suivant:

- les éléments z de l'ensemble sont stockés par les boîtes notées $\boxed{<?}$, $\boxed{=?}$ et $\boxed{>?}$ présentes aux différents nœuds d'une structure arborescente formée par le réseau lui-même.
- chaque nœud, symbolisé par les grands trapèzes en pointillé est constitué des fonctions réalisant les opérations *existe* [fig 3.17], *inclue* [fig 3.19], *retire* etc., pour un élément z, ainsi que des connections vers les sous-ensembles composés des éléments inférieurs à z d'une part (petit trapèze noté $< z$) et supérieur à z d'autre part (trapèze $> z$).
- l'opération *existe*(x, e) déclenchée par l'arrivée de x sur l'entrée du réseau 3.17 se déroule de la manière suivante:

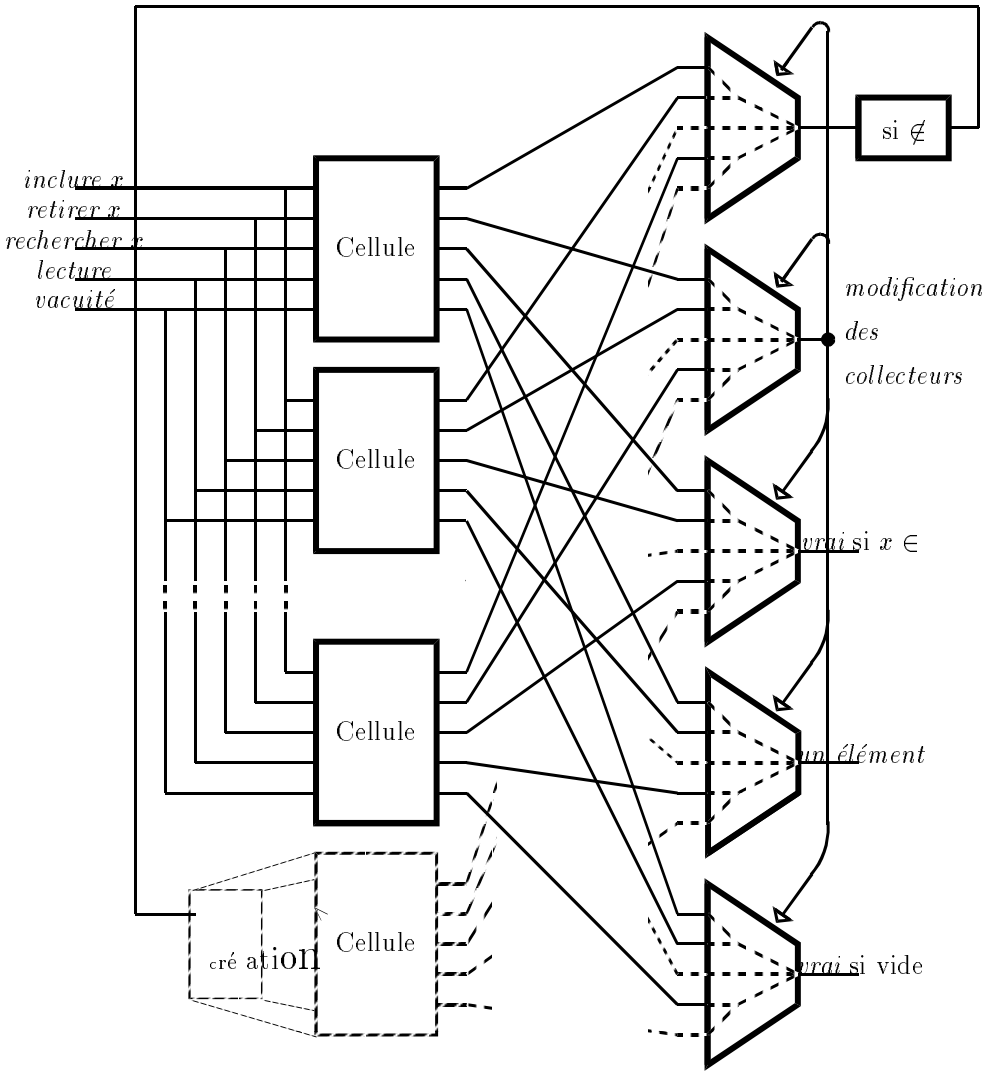


Figure 3.15: Implantation d'un ensemble par des boîtes parallèles. L'arrivée d'une valeur x sur l'une des entrées *insertion*, *suppression* ou *existence*, ou d'une demande *test de vacuité* ou *élément*, active tous les réseaux, représenté figure 3.16 contenant chacun un des élémemnts de l'ensemble. Les réponses de tous ces réseaux sont collectées par des collecteurs arborescents. L'insertion d'un nouvel élément n'est faite que s'il n'existait pas; il y a alors création d'un nouveau réseau;

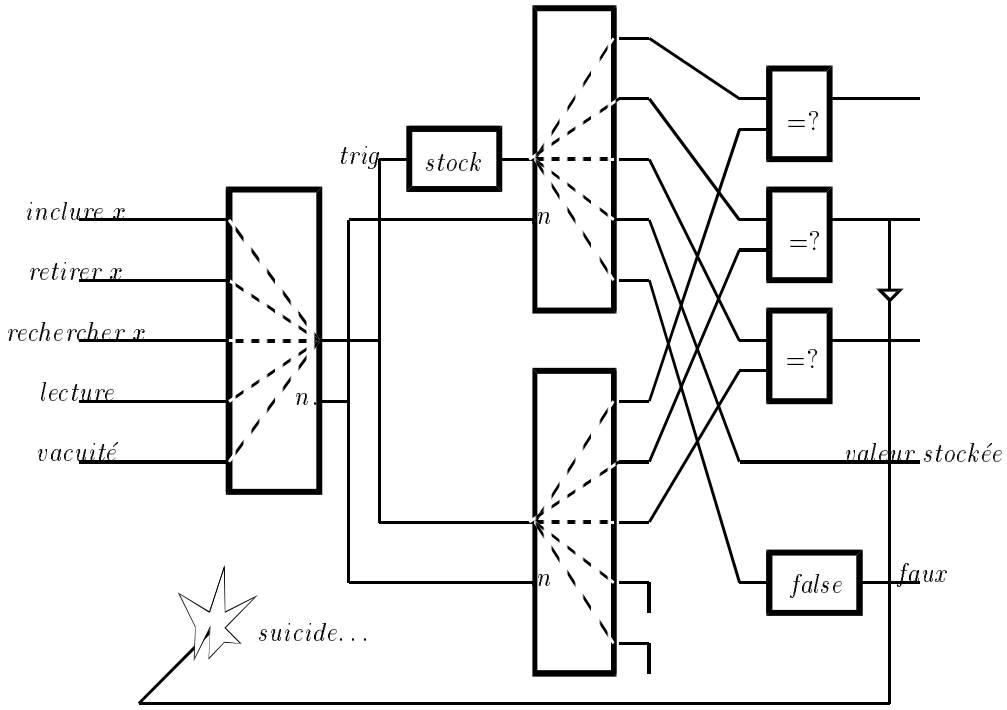


Figure 3.16: Portion du réseau de la figure 3.15 correspondant à une cellule. La boîte *stock* contient un des éléments de l'ensemble. L'arrivée d'un x, égal à cette valeur stockée, sur l'entrée *retire* provoque la disparition du réseau.

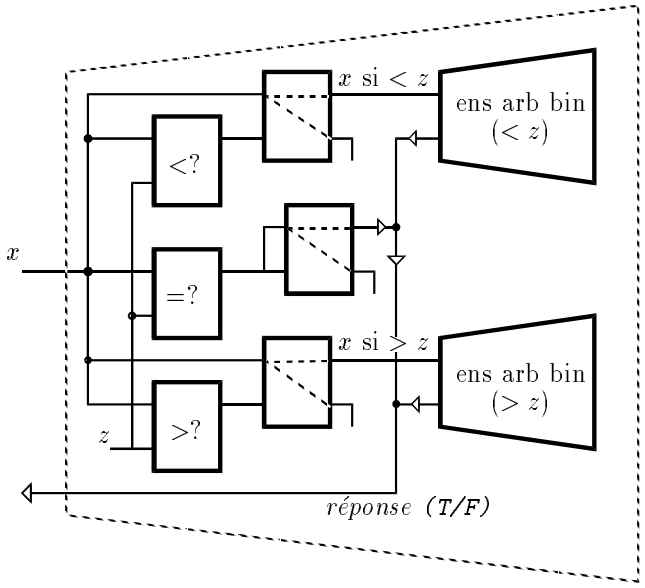


Figure 3.17: Ensemble implanté par un arbre binaire. Test de l'existence d'une valeur x. La racine contient l'élément z, la réponse *vrai* ou *faux* est renvoyée directement au réseau interrogateur.

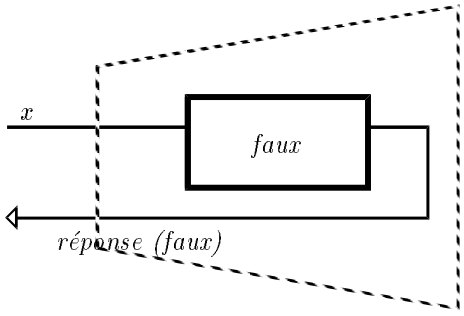


Figure 3.18: Ensemble vide. Test de l'existence d'une valeur x. La réponse, *faux*, est renvoyée directement au réseau interrogateur.

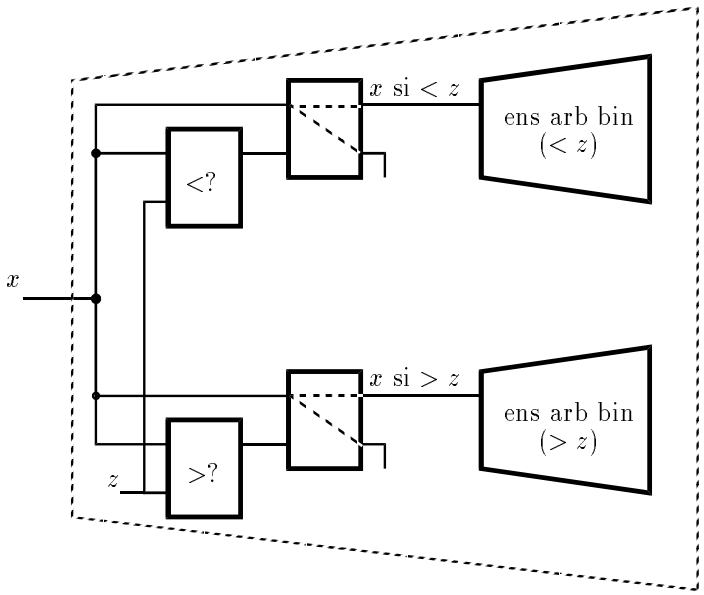


Figure 3.19: Ensemble implanté par un arbre binaire. Inclusion d'une valeur x. La racine contient l'élément z, si $x = z$ il ne se passe rien, sinon x est inclu dans l'arbre ($<$) ou ($>$).

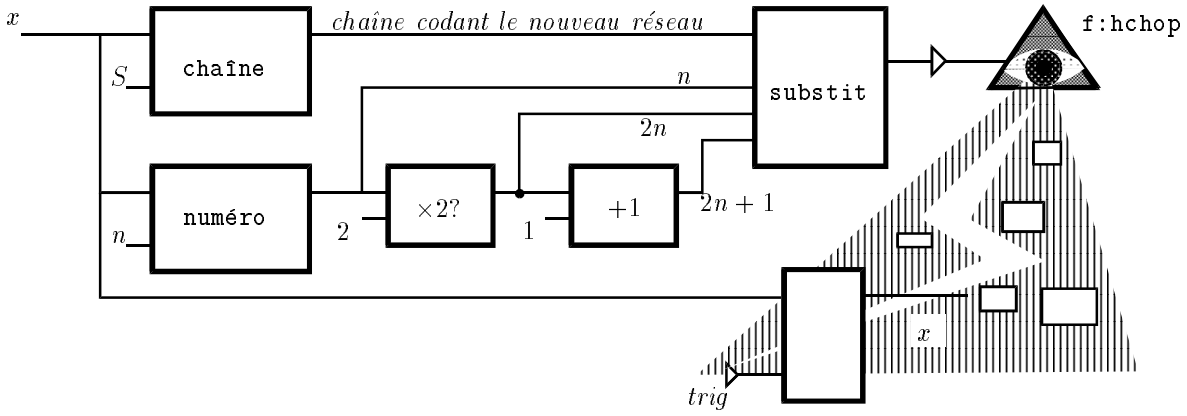


Figure 3.20: Inclusion d'une valeur x dans un ensemble vide. Il y a création de nouvelles boîtes (par la fonction `f:hchop`) correspondants aux réseaux des figures 3.17, 3.18, 3.19 et 3.20. La chaîne de caractères S est la codification de ces boîtes.

- si $x = z$, une réponse *vrai* est renvoyée par l'aiguillage connecté à la boîte $\boxed{=?}$;
- si $x < z$, le sous-réseau correspondant $e^<$ est activé, sa réponse *vrai* ou *faux* est transmise en retour.
- si $x > z$, le sous-réseau $e^>$ est activé.
- si l'ensemble est vide, la réponse *faux* est immédiatement renvoyée par le réseau 3.18 qui code l'ensemble vide.

• l'opération *inclue*(x, e) déclenchée par l'arrivé de x sur l'entrée du réseau 3.19 se déroule de la manière suivante:

- si $x = z$, rien ne se passe, un élément n'étant pas inclus deux fois.
- si $x < z$ il sera inclus dans le sous-ensemble $e^<$ composé des éléments $< z$.
- si $x > z$ il sera inclus dans $e^>$.
- Si l'ensemble est vide il y a création de nouvelles boîtes. Nous présentons en détail la manière dont cela peut être réalisé dans le cas du *PS300*:

  Les commandes PSDDNL, codant pour tous les différents réseaux qui sont représentés ici, sont envoyées à l'interpréteur de commandes, noté `f:hchop`, chargé de créer les nouvelles boîtes.

  Ces commandes sont construites à partir de la chaîne de caractères S, véritable information génétique, qui doit être mise à jour afin de correspondre à la codification du nœud à créer. En particulier toutes les boîtes doivent avoir des noms différents: il est possible de les numéroter en conséquence. Par exemple les boîtes filles d'un nœud numéroté n sont affectés des numéros $2n$ et $2n + 1$, si la toute première boîte est numérotée 1, ses filles 10_2 et 11_2, leurs filles 100_2, 101_2, 110_2 et 111_2 etc., tous les numéros seront différents.

  La chaîne S code également les nouvelles connections ainsi que les ordres de destruction du réseau *feuille* de la figure 3.20 qui sera recréé à l'identique en double exemplaire pour les réseaux fils (pour les prochaines inclusions).

- l'opération *retire(x, e)* est plus délicate puisqu'il est nécessaire de modifier les connections entre les différents nœuds, en effet le réseau doit réaliser la fonction

  retire(x,e)=
  si vide(e) **retourne**(e)
  si (x <rac(e)) enrac(retire($x, e^<$),rac(e),$e^>$)
  si (x >rac(e)) enrac($e^<$,rac(e),retire($x, e^>$))
  si (x =rac(e)) enrac($e^<$,mini($e^>$)) ,retire($e^<$,mini($e^>$))

  où mini(a) recherche le plus petit élément de l'arbre (la feuille la plus à gauche puisque a est trié).

3.4.2 Autres structures de données

Arbres et graphes

Des structures plus compliquées telles que des arbres ou des graphes peuvent être construites. Comme pour les ensembles, leur réseau peut correspondre à la structure de données ou la simuler. Ainsi, le graphe associé au réseau peut être assimilé au graphe que l'on veut construire, les connections entre les boîtes réalisant les arcs.

Liste, tables et tableaux

Remarquons que, contrairement à la notion de pile, comme nous l'avons dit au paragraphe 3.3.5, la structure de *liste séquentielle* est particulièrement bien adaptée à la programmation cadencée par les données, elle est en fait supportée naturellement par les files d'attente des entrées de boîtes. Par analogie, la notion de tableaux est relativement naturelle en programmation classique du fait de la configuration linéaire de la mémoire. Une implantation possible d'un tableau peut être réalisée par une liste de couples (indice,valeur), ou mieux, par adressage direct, si un élément de réseau est créé pour chaque valeur stockée.

3.4.3 Gestion des structures de données communes

L'accès à une tructure de données par plusieurs réseaux pose des problèmes de partage de ressources et d'efficacité. Le principe même de ces structures uniques, évoluant au cours du temps est contraire à la notion de cadencement par les données et ne peut être résolu élégamment que par des techniques plus sophistiquées citées ci-dessous, en particulier l'utilisation de données labelées [Gaudiot 1985].

3.5 Implantation réelle des réseaux

Les réseaux de fonctions cadencés par les données dont nous venons de donner une description logique doivent être implantés physiquement sur un ou plusieurs ordinateurs mono ou multi-processeurs. La réalisation pratique de l'implantation dépend du langage, et des machines cibles. Nous reviendrons plus loin sur celle du langage PSDDNL de la station graphique *PS300* Evans & Sutherland.

Différents modes de fonctionnement sont envisageables:

- Les entrées de boîtes sont organisées en files d'attente infinies, dans ce cas les fonctions peuvent expédier, vers chaque destinataire, leurs résultats au fur et à mesure de leur production.

- Si sur un fil ne peut être stockée qu'une ou peu de données, les fonctions devront elles-mêmes gérer leurs émissions et éventuellement ne s'exécuter que si cette émission devient possible.

- Pour contourner cette difficulté, on peut imaginer une exécution de sur demande du receveur *(Demand Driven Network*, par opposition à *Data Driven)*.

- A chaque donnée issue d'une fonction peut être associé un jeton spécifiant l'expéditeur, le destinataire et le flux auquel il correspond. Si une fonction s'exécute lorsque toutes ses entrées associées à un flux donné sont présentes, il est possible , dans ce cas, de mélanger et même d'intervertir les différents pas des boucles itératives lorsque ceux-ci sont indépendants. Cette technique est en particulier employée dans *Id*, *Val* [Arwind *et al.*, 1978, Arwind et Gostelow, 1982, Ackerman et Dennis, 1979, Watson et Gurd, 1982].

3.6 La programmation du *PS300*

Le *PS300* Evans & Sutherland est à notre connaissance la seule machine commercialisée dont la programmation soit uniquement basée sur le modèle du cadencement par les données. Le seul langage disponible est le langage PSDDNL, dérivé des langages *Data Driven* de A.L. Davis [Davis 1978]; il est décrit précisement en annexe A. Aucun autre moyen de programmation n'est en effet possible, même le système d'exploitation est cadencé par les données.

3.6.1 Spécificité du langage

Nous signalons en particulier que

- Les entrées des fonctions sont des files d'attente infinies.
- Les connections multiples sur une même entrées sont autorisées. La précautions à prendre pour éviter les éventuelles collisions incombent au programmeur. Celui-ci peut décider que les cas de non-déterminisme n'arrivent jamais, à condition que l'utilisateur interagit *normalement*, sans faire d'erreurs de manipulation, ou contruire un réseau sûr en utilisant, par exemple, des sélecteurs et des boîtes de synchronisation.
- De là découle qu'un nombre indéterminé de sorties peuvent être connectées à une même entrée, et que les collecteurs arborescents ne sont plus toujours nécessaires.
- Les fonctions de base du langage sont relativement nombreuses et de haut niveau (voir la liste en annexe). Elles sont très bien choisies et présentent peu de lacunes dans le domaine graphique (difficulté du calcul de l'arc tangente, par exemple). De plus l'écriture, sur un ordinateur hôte, en *Pascal* ou en *Fortran* de nouvelles fonctions est maintenant possible; nous ne l'avons pas encore expérimenté.
- Le réseau de fonctions cadencé par les données n'existe pas phyiquement mais est simulé par le processeur MC68000. Les fonctions de base sont telles que leur temps d'exécution n'excède jamais 200 ms, par ailleurs l'algorithme d'activation des boîtes prêtes à être exécutées garantit un non-blocage du réseau. Une erreur

dans la programmation d'un réseau peut néanmoins provoquer l'accumulation de données sur une entrée de boîte et occasionner des remplissages de la mémoire et un donc arrêt du processeur.

Nous regrettons que

- Il n'est pas possible de créer de nouvelles fonctions par encapsulation de fonctions existantes, la programmation structurée et hiérarchisée de réseaux devient difficile et ne peut finalement être faite par le programmeur que par des artifices de dénomination de boîtes lors de l'écriture du programme. Nous y reviendrons au paragraphe 3.6.2.
- Le réseau n'est pas réparti sur une ensemble de processeurs rapides. Ceci permettrait l'écriture de très gros programmes qui pourraient être particulièrement intéressants pour la manipulation interactive de la structure de données graphique.

3.6.2 Méthode de programmation du *PS300*

Comme nous l'avons déjà souligné en 3.3.6, la définition d'un réseau cadencé par les données par le biais d'une notation fonctionnelle nécessite l'emploi d'un langage approprié.

Le langage PSDDNL se veut *plus proche des réseaux* et n'est pas destiné à cela. Dans cet esprit, il est possible d'aborder la programmation d'un réseau de multiples façons:

Au pas à pas

La conception d'un réseau peut être entreprise sans en avoir défini une vue d'ensemble, il suffit de commencer par dessiner une boîte puis les boîtes vers lesquelles devront être connectées ses sorties, etc.; les fonctions nécessaires à la création d'une interface de manipulation d'objets graphiques n'étant pas trop compliquées, on arrive après beaucoup d'essais-erreurs à obtenir le réseau souhaité. La réalisation d'une autre fonction se fait de la même manière, sans pouvoir pourtant réutiliser ce qui a déjà été fait (les noms de fonctions devant être différents).

La complexité d'un réseau est souvent due en grande partie à sa taille, *A*GRAPH, par exemple, a nécessité l'écriture de presque 2000 lignes de programmes. Il est apparu rapidement qu'il était vital de programmer de façon modulaire, certaines fonctions étant souvent répétées, au sein de la même application mais aussi dans des applications distinctes, leur assemblage étant dans ce cas généralement différent.

Par modules

Nous avons introduit la programmation par modules, chacun étant assimilable à une carte électronique, ces cartes étant reliées les unes aux autres par des *connecteurs plats* qu'il suffit de brancher par une suite de commandes rappelant dans un certain sens une édition de liens (ceci correspond en fait au transfert de paramètres par rang). En annexe figurent les instructions concernant la programmation des réseaux de manipulation de la caméra virtuelle dans *A*GRAPH. Les modules sont schématisés sur la figure C.1.

Avec boîtes et superboîtes

Cette idée de modules serait facilement adoptée s'il existait dans le langage de programmation une notion de **superboîte**, c'est-à-dire la possibilité de créer simplement de nouvelles fonctions (ou boîtes) par assemblage de fonctions existantes. Cette possibilité n'existe malheureusement pas dans le langage tel qu'il est proposé, sa mise en œuvre ne devrait guère poser de problèmes, la principale difficulté étant due à la nécessité d'écriture d'un analyseur syntaxique pour le langage ainsi étendu. Les programmes seraient naturellement structurés.

Avec représentation graphique

Puisqu'un réseau peut être assimilé à un circuit électronique, représenté par un dessin bidimensionnel des boîtes et de leurs connections, il est évident qu'un logiciel permettant la visualisation graphique du réseau ainsi que la construction interactive de celui-ci faciliterait beaucoup la tâche du programmeur. Ce logiciel existe, il est distribué par la société Evans & Sutherland et peut tourner sur une station graphique

PS300. Malheureusement nous ne l'utilisons pas au laboratoire, l'immobilisation d'une telle machine à des fins de développement de programmes étant diffilement justifiable.

La représentation graphique pourrait être alliée à la possibilité d'utilisation de *superboîtes*, la visualisation d'un réseau devrait se faire de manière hiérachisée, en respectant les imbrication des différentes *superboîtes*, le niveau de détail étant fonction de l'étendue du réseau affiché.

Analyse descendante et ascendante

Tout ceci est également valable lors de l'écriture du réseau, sa construction pouvant être effectuée par analyse descendante ou ascendante, les *superboîtes* être utilisées avant d'être définies ou inversement, un réseau existant devenir le modèle d'une nouvelle *superboîte*.

Aide à la mise au point

Lors de la mise au point d'un programme, il est souvent nécessaire de visualiser les informations circulant entre les boîtes. Il est facile de concevoir l'équivalent d'un **analyseur logique** pour circuits électroniques. Celui-ci consiste en une sorte de pince qu'il suffit de positionner sur les broches du circuit intégré et qui permet de visualiser sur un écran les signaux reçus et émis par le circuit. Ceci peut se faire exactement de la même manière pour les réseaux *Data Driven*, l'affichage des données véhiculées pouvant se faire en temps réel, avec une éventuelle *bufférisation*, l'insertion de boîtes de synchronisation permettant l'exécution *pas à pas*.

Outils d'aide à la programmation

Ces différents aspects mériteraient d'être effectivement concrétisés par la réalisation des outils adéquats. La mise au point de grands programmes cadencés par les données un tant soit peu complexes ne peut se concevoir sans un environnement de programmation adapté.

3.7 Les réseaux et *A*Graph

Le logiciel *A*GRAPH a nécessité l'écriture de plus de 2000 lignes de programmes décrivant un réseau de fonctions cadencé par les données.

3.7.1 Organisation générale

Ce réseau est organisé comme le logiciel, dont les principes sont décrits au chapitre 2. La figure 2.4 présente le schéma général:

- le **réalisateur** aiguille les commandes (touches et boutons) de l'utilisateur vers:
  - le **metteur en scène** qui dirige les acteurs et décide de l'envoie vers l'ordinateur hôte d'une scène clé, son réseau est représenté figure 2.2,
  - l'**animateur** qui transmet régulièrement aux acteurs les paramètres correspondant à leur rôle,
  - le **caméraman** qui commande la caméra en fonction des ordres de l'utilisateur. Certains modules sont présentés en détail dans l'annexe C.
- d'autres réseaux permettent des échanges avec l'ordinateur hôte, la visualisation des paramètres de la caméra et la gestion de la caméra 16mm ou vidéo qui éventuellement filme l'écran

3.7.2 Acteur, rôle, mémorisation des scènes clés

La structure de données graphique d'un acteur, défini précisément en 2.4.2, est mise à jour par le réseau représenté figure 3.22, en fonction des informations provenant soit du metteur en scène (fig. 2.2), soit du réseau *rôle* (fig. 3.23) de l'animateur comme le montre la figure 3.21. La mémorisation des paramètres d'un acteur est faite par le réseau `CLIC` présenté figure 3.24.

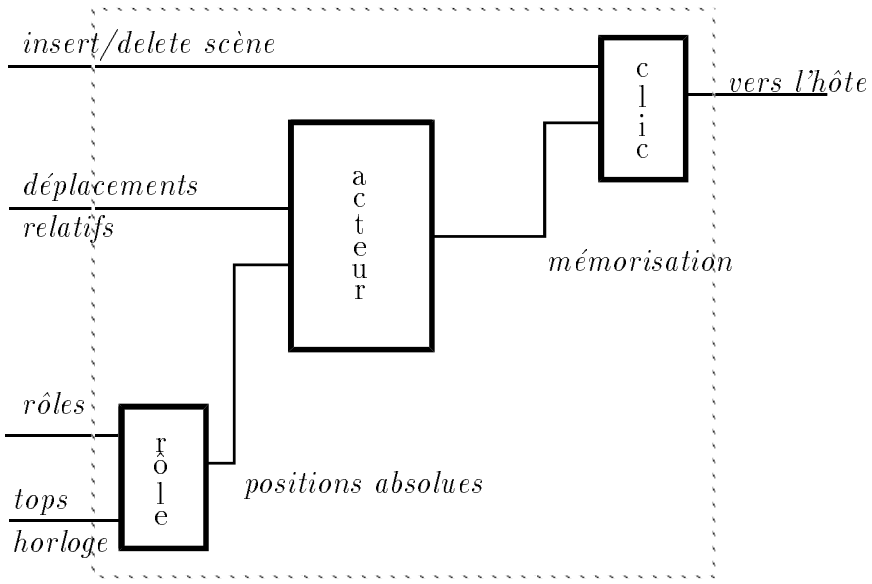


Figure 3.21: L'acteur, son rôle et son réseau de mémorisation de position clé (clic). Le metteur en scène peut modifier chaque paramètre de l'acteur à l'aide des *dials*. Il définit une scène clé à l'aide de la commande *insert* ou la supprime par *delete*. Les différentes positions de l'acteur constituant son rôle lui sont affectées sur commande de l'animateur (tops horloge).

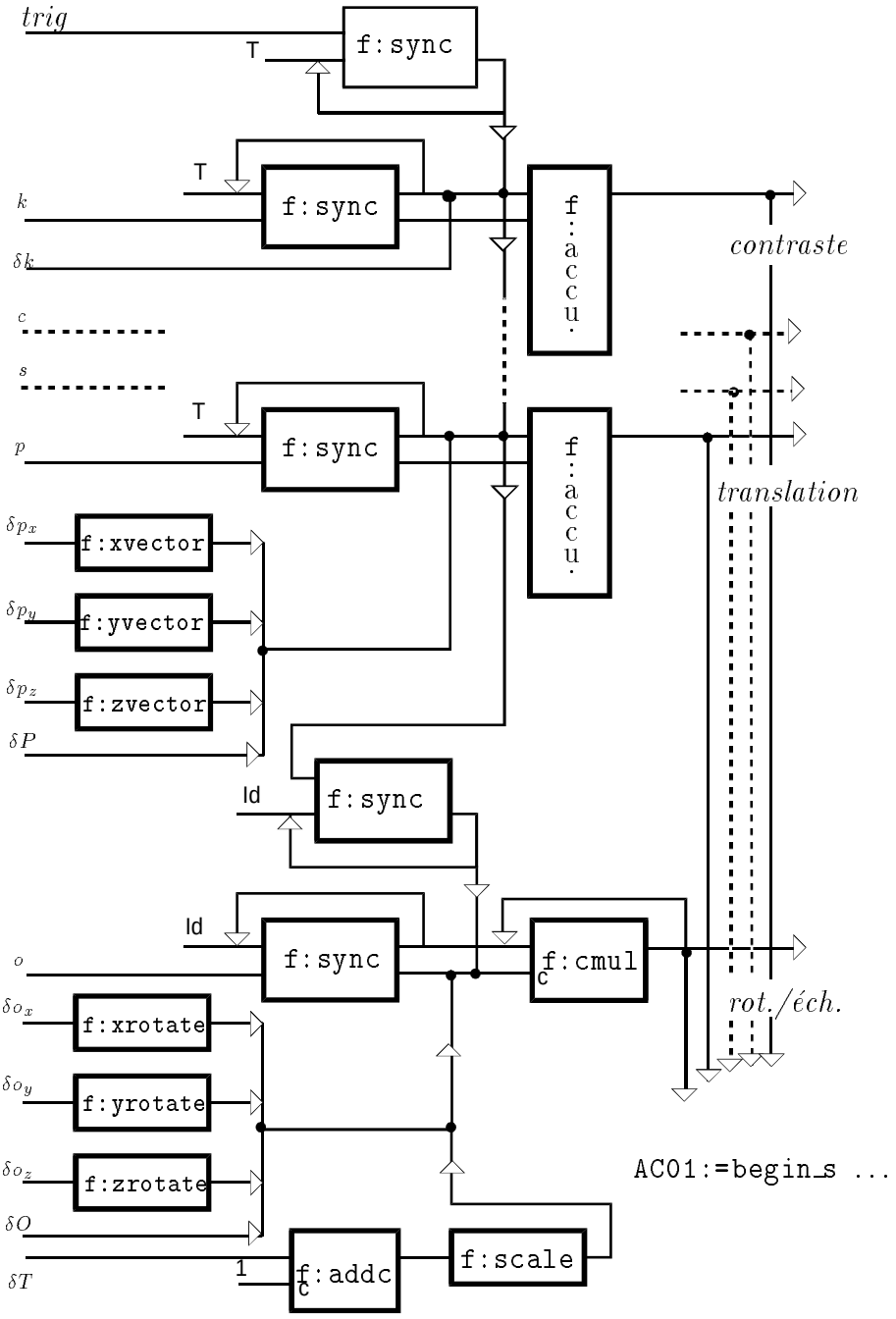


Figure 3.22: Les paramètres d'un acteur peuvent être modifiés en valeur relative par l'intermédiaires des *dials* ou en valeurs absolues. Les nouveaux paramètres sont envoyés au réseau `CLIC` représenté figure 3.24 et permettent la mise à jour de l'instruction `AC01:=begin_s` ... de la base de données graphique.

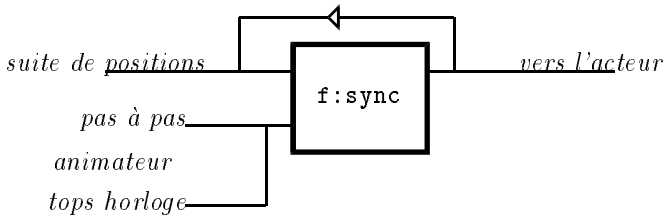


Figure 3.23: Réseau de stockage d'une suite de positions constituant le rôle de l'acteur (ce peut être une suite de matrices d'orientation, de points de la trajectoire tridimensionnelle, ou des valeurs de couleur par exemple). L'arrivée d'une commande d'avance en *pas à pas* ou en mode animation automatique déclenche la mise à jour du paramètre de l'acteur.

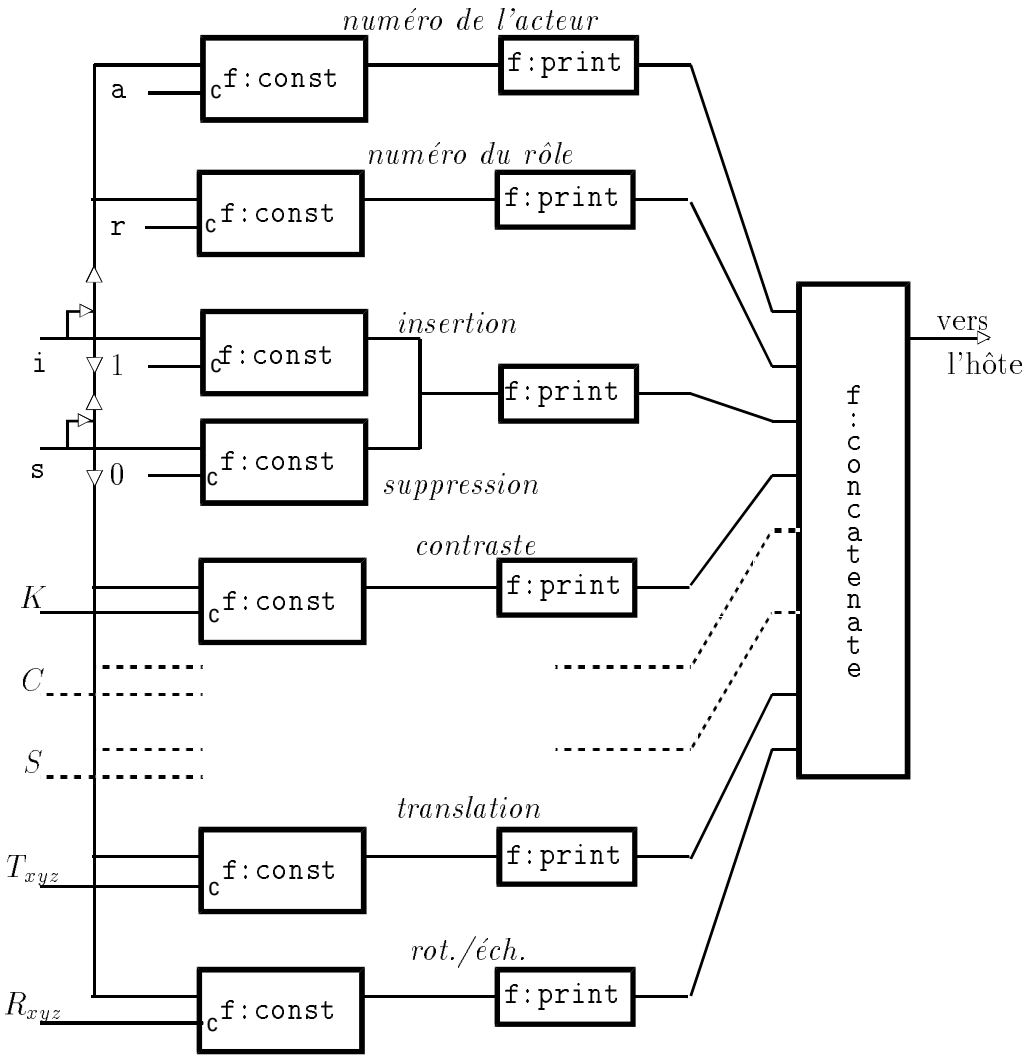


Figure 3.24: CLIC: Ce réseau mémorise tous les paramètres d'un acteur (K, C, S, T_{xyz} et R_{xyz}). Les commandes d'*insertion* (`i`) ou de *suppression* (`s`) de clichés provoquent la conversion en chaînes de caractères, la concaténation et l'émission vers l'ordinateur hôte. Dans le cliché ainsi créé figurent, en plus des paramètres, le numéro (`a`) de l'acteur, le numéro (`r`) du rôle lors du cliché (équivalent à une date) et la valeur 1/0 codant l'*insertion* ou la *suppression*.

Chapitre 4

Interpolation

L'utilisateur *metteur en scène* doit pouvoir définir les mouvements d'un acteur par la donnée d'un petit nombre de positions et d'orientations à des moments quelconques. Avec cela, la machine doit calculer une trajectoire et une suite de rotations qui paraîtront *naturelles* ou tout au moins *agréables à regarder* lors de l'animation. Si aucune prise en compte de lois physiques ou d'équations de mouvements n'est faite, des critères de continuité et de mouvements lisses et progressifs semble être le plus appropriés. C'est cette solution que nous avons choisie pour l'ensemble des paramètres dont dépend l'animation.

4.1 Introduction

Comment définir simplement une courbe ou une trajectoire autrement que par une équation ou la donnée explicite de chacun de ses points?

Lorsque la courbe est continue et *lisse* elle peut être approximée plus ou moins précisement par une ou plusieurs fonctions simples. Celles-ci peuvent être des polynômes, des combinaisons de fonctions trigonométriques, des suites de segments de droite ou d'arcs de cercle, etc. [Bezier 1987].

Pour réaliser nos séquences d'animation, nous nous intéresserons particulièrement au cas où les courbes que l'on cherche à approximer sont données par une suite de points par lesquels elles doivent passer. Ces courbes peuvent être des trajectoires dans ^{3}, des grandeurs scalaires ou la représentation d'une rotation. Nous les consid-

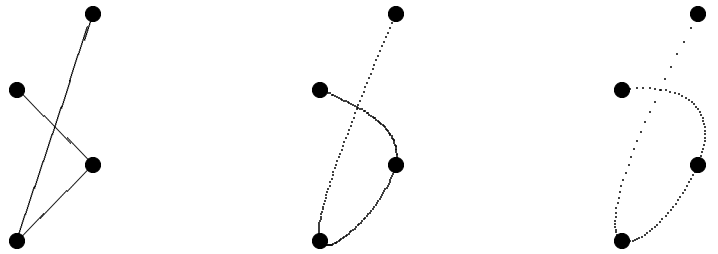


Figure 4.1: Trois courbes interpolant les points p_0, p_1, p_2, p_3.

érons de manière générale comme des fonctions de dans ^{n} et les définissons comme des courbes paramétriques.

Nous étudierons spécialement les techniques d'interpolation par fonctions *splines*, où les fonctions d'interpolation sont des polynômes de faible degré, essentiellement de degré 3. Nous n'aborderons pas du tout les méthode générales d'approximation [Mortenson 1985, Piegl 1987], considérant que les points donnés par l'utilisateur sont en fait des points de *rendez-vous* par lesquels les "courbes" doivent passer.

4.2 Courbes paramétriques

Une courbe paramétrique dans l'espace ^{3}, ou d'une manière générale ^{n}, peut être définie comme étant l'ensemble des points dont les coordonnées (p_x, p_y, p_z) sont données par les fonctions de dans dépendant du même paramètre u de la forme

$$p_x = x(u) \qquad p_y = y(u) \qquad p_z = z(u)$$

Si u décrit un intervalle $[a, b]$, $p(u)$ décrit un segment de courbe.

Nous déterminerons les équations paramétriques des courbes passant par deux ou quatre points qui sont tangentes à des directions données, puis nous définirons des suites de telles courbes afin d'approximer des fonctions spécifiées par n points respectant certaines conditions de continuité et de dérivabilité.

4.2.1 Deux points, deux tangentes

Conditions à respecter

Soient deux points p_0 et p_1 et deux vecteurs t_0 et t_1, nous allons construire le segment de courbe paramétrique p définie sur $[0,1]$ vérifiant les conditions:

1. p est continu sur $[0,1]$
2. $p(0) = p_0$
3. $p(1) = p_1$
4. p est tangent en p_0 à un vecteur t_0
5. p est tangent en p_1 à un vecteur t_1

Il existe évidemment une infinité de courbes vérifiant ces conditions. Nous nous resteindrons à celles dont les équations paramétriques sont des polynômes en u (remarquons que de ce fait, la condition de continuité est vérifiée).

Degrés de liberté

Plaçons-nous dans ^{n}. La position d'un point dépend de n paramètres, celle d'un vecteur tangent de $n-1$ (en effet, la direction d'un vecteur peut être définie, par exemple, par la donnée d'un point de la sphère unité; ses coordonnées ne sont pas indépendantes: lorsqu'on en connait $n-1$, on peut calculer la n^{eme}, la somme des carrés étant égale à 1). Notre courbe dépend donc de $2(n+n-1)$ soit $4n-2$ paramètres indépendants. Si $x(u)$, $y(u)$ et $z(u)$ sont des polynômes de degré k, chacun aura $k+1$ degrés de liberté (autant qu'il y a de coefficients à déterminer). Il faut donc choisir k tel que $n(k+1) \geq 4n-2$, c'est-à-dire $k \geq 3-2/n$, soit

$$k \geq 2 \quad dans \ ^2$$

ou

$$k \geq 3 \quad dans \ ^n \quad \forall n \geq 3.$$

Dans le plan, les polynômes de degrés 2 sont justes suffisants. Dans ^{n}, des polynômes de degré 3 et plus suffisent dans tous les cas. Nous traiterons uniquement des polynômes cubiques.

Détermination des coefficients des polynômes cubiques

Posons

$$\begin{aligned} x(u) &= a_{3_x}u^3 + a_{2_x}u^2 + a_{1_x}u + a_{0_x} \\ y(u) &= a_{3_y}u^3 + a_{2_y}u^2 + a_{1_y}u + a_{0_y} \\ z(u) &= a_{3_z}u^3 + a_{2_z}u^2 + a_{1_z}u + a_{0_z} \end{aligned}$$

que nous écrivons matriciellement

$$p(u) = AU$$

avec

$$A = \begin{pmatrix} a_{3_x} & a_{2_x} & a_{1_x} & a_{0_x} \\ a_{3_y} & a_{2_y} & a_{1_y} & a_{0_y} \\ a_{3_z} & a_{2_z} & a_{1_z} & a_{0_z} \end{pmatrix}, \qquad U = \begin{pmatrix} u^3 \\ u^2 \\ u \\ 1 \end{pmatrix}.$$

Les conditions 1 et 2 s'expriment alors

$$A \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = p_0, \qquad A \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = p_1$$

et puisque

$$\frac{dAU}{du} = A\frac{dU}{du} = A \begin{pmatrix} 3u^2 \\ 2u \\ 1 \\ 0 \end{pmatrix}$$

les conditions aux tangentes s'écrivent

$$A \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = t_0 \qquad et \qquad A \begin{pmatrix} 3 \\ 2 \\ 1 \\ 0 \end{pmatrix} = t_1.$$

Tout cela peut s'écrire matriciellement

$$A \begin{pmatrix} 0 & 1 & 0 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix} = AM = \begin{bmatrix} p_0 & p_1 & t_0 & t_1 \end{bmatrix},$$

$\begin{bmatrix} p_0 & p_1 & t_0 & t_1 \end{bmatrix}$ étant une matrice 3×4 construite à partir des 4 vecteurs colonnes.

La matrice M est inversible, son inverse est

$$M^{-1} = \begin{pmatrix} 2 & -3 & 0 & 1 \\ -2 & 3 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 1 & -1 & 0 & 0 \end{pmatrix}$$

et on obtient les 12 coefficients par

$$A = \begin{bmatrix} p_0 & p_1 & t_0 & t_1 \end{bmatrix} M^{-1}$$

La figure 4.2 correspond aux courbes dont les tangentes forment des angles de 0° à 360° en p_0 et 60° en p_1.

Choix du module des tangentes

Attention, la solution n'est pas unique car les vecteurs t_0 et t_1 sont définis chacun à une constante près. On peut en effet écrire

$$t_0 = \alpha_0 \tau_0 \qquad et \qquad t_1 = \alpha_1 \tau_1$$

où τ_0 et τ_1 sont des vecteurs unitaires indiquant la direction des tangentes.

Les choix de α_0 et α_1 sont arbitraires et déterminent la forme *plus ou moins arrondie* de la courbe [fig 4.3], des conditions supplémentaires (rayon de courbure, dérivée seconde) seraient à préciser. Ces coefficients traduisent l'influence des tangentes sur l'allure de la courbe, de petites valeurs raidissent le tracé, de grandes valeurs peuvent amener à de somptueux détours.

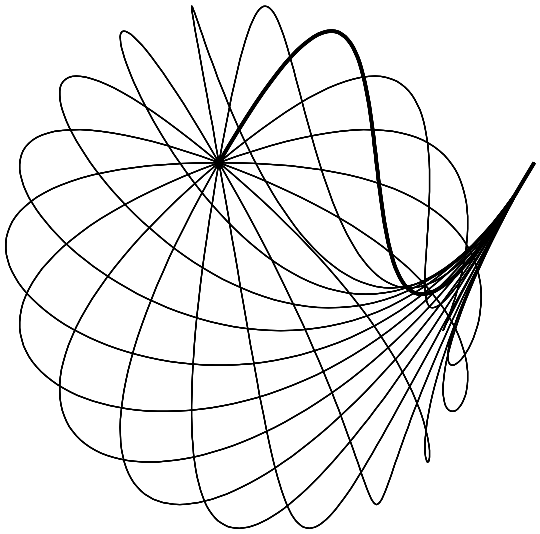


Figure 4.2: Courbes polynômiales cubiques entre deux points, avec différents angles de départ. En gras figure la courbe correspondant à un angle de 60°.

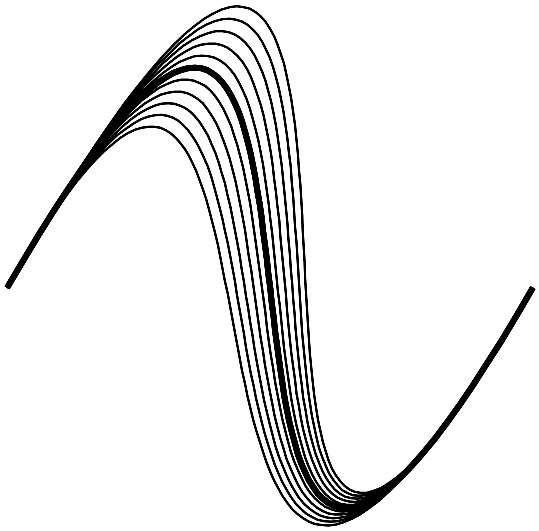


Figure 4.3: Courbes polynômiales cubiques entre deux points, avec différents modules de la tangentes de départ. En gras figure la courbe correspondant à un module égal à 5 fois la distance entre les deux points.

Comment minimiser les courbures?

Nous allons déterminer *une* condition qui permette de choisir les valeurs α_0 et α_1 qui *minimisent les courbures*. A priori nous ne savons pas quel sens donner à cette notion, s'agit-il de minimiser la courbure maximale, d'obtenir la moyenne la plus petite possible, etc.?

Nous choisirons de minimiser l'intégrale sur toute la courbe de la dérivée seconde; pour des raisons de symétrie, nous prendrons $\alpha_0 = \alpha_1 = \alpha$.

Il s'agit alors de trouver α tel que

$$F(\alpha) = \int_0^1 \| \frac{d^2 p(u)}{du^2} \|^2 \, du \quad soit\ minimal.$$

Calculons cette intégrale pour chacune des composantes (en omettant les indices)

$$\begin{aligned} \int_0^1 \left(\frac{d^2 p(u)}{du^2}\right)^2 du &= \int_0^1 (6a_3 u + 2a_2)^2 \, du \\ &= [12a_3^2 u^3 + 12a_3 a_2 u^2 + 4a_2^2 u]_0^1 \\ &= 12a_3^2 + 12a_3 a_2 + 4a_2^2. \end{aligned}$$

En exprimant les a_i en fonction de p_0, p_1, t_0 et t_1 on a finalement

$$\int_0^1 \left(\frac{d^2 p(u)}{du^2}\right)^2 du = 4[t_0^2 + t_1^2 + t_0 t_1 - 3(p_1 - p_0)(t_0 + t_1) + 3(p_1 - p_0)^2]$$

Posons $t_0 = \alpha\tau_0$, $t_1 = \alpha\tau_1$ et $P = p_1 - p_0$.

Toutes ces grandeurs sont des vecteurs à n dimensions, on obtient en sommant sur les n composantes

$$F(\alpha) = 4 \sum_{i=1}^{n} [(\tau_{0i}^2 + \tau_{1i}^2 + \tau_{0i}\tau_{1i})\alpha^2 - 3P_i(\tau_{0i} + \tau_{1i})\alpha + 3P_i^2].$$

Les vecteurs τ_0 et τ_1 étant normés, en notant θ l'angle que font entre elles les deux tangentes, ϕ_0 et ϕ_1 les angles que font les tangentes avec la direction $p_0 p_1$ on a

$$\sum_{i=1}^{n} \tau_{0i}^2 = 1 \qquad \sum_{i=1}^{n} \tau_{1i}^2 = 1 \qquad \sum_{i=1}^{n} \tau_{0i}\tau_{1i} = \cos\theta$$

et

$$\sum_{i=1}^{n} P_i \tau_{0i} = \cos\phi_0 \|P\| \qquad \sum_{i=1}^{n} P_i \tau_{1i} = \cos\phi_1 \|P\|.$$

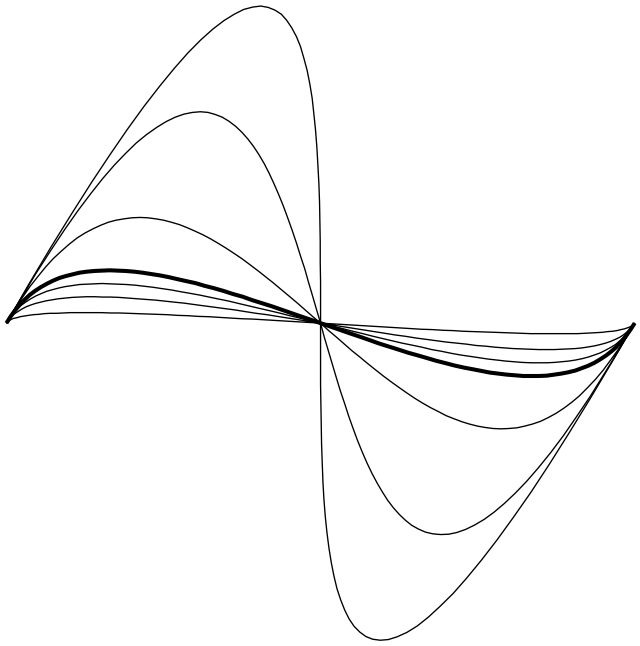


Figure 4.4: Influence de la valeur de α, les tangentes font un angle de 60°. En gras figure la courbe pour α du minimum, les autres courbes correspondant, dans l'ordre de la plus raide à la plus ronde, à des valeurs 0.2α, 0.5α, 0.75α, 2α, 4α et 6α.

D'où

$$F(\alpha) = 4(2 + \cos\theta)\alpha^2 - 12(\cos\phi_0 + \cos\phi_1)\|P\|\alpha + 12\|P\|^2.$$

Cette fonction atteint son minimum pour

$$\alpha = \frac{3(\cos\phi_0 + \cos\phi_1)}{(2 + \cos\theta)}\|P\|.$$

Les figures 4.4 et 4.5 nous fournissent quelques exemples de tracés en fonction de α.

Dans certains cas, non souhaités, on a $\alpha = 0$; la courbe est alors réduite au segment joignant les deux points.

4.2.2 Quatre points

Nous venons de construire la courbe paramétrique passant par deux points, tangente à deux vecteurs [4.2.1], nous allons maintenant définir la courbe paramétrique cubique passant par 4 points.

Pour cela il suffit d'attribuer 4 valeurs u_i au paramètre u telles que

$$p(u_i) = p_i \quad i = 0, 1, 2, 3.$$

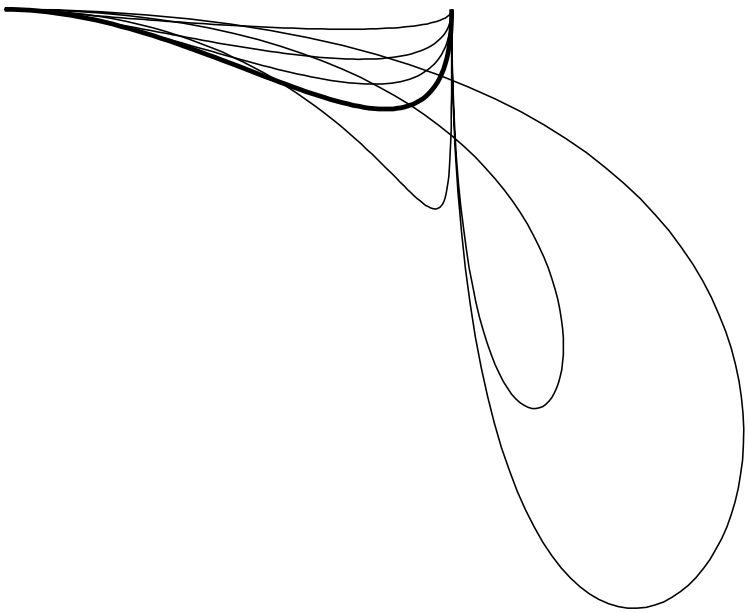


Figure 4.5: Influence de la valeur de α. Les tangentes font des angles de 0 et 90°. En gras figure la courbe pour α du minimum, les autres courbes correspondant, de la plus raide à la plus ronde, à des valeurs 0.2α, 0.5α, 0.75α, 2α, 4α et 6α.

Nous choisirons évidemment $u_0 = 0$, $u_3 = 1$. Le choix de u_1 et u_2 pourra dépendre de conditions extérieures.

La matrice A se calcule par résolution du système

$$A \begin{pmatrix} 0 & u_1^3 & u_2^3 & 1 \\ 0 & u_1^2 & u_2^2 & 1 \\ 0 & u_1 & u_2 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} = \begin{bmatrix} p_0 & p_1 & p_2 & p_3 \end{bmatrix}.$$

La figure 4.6 montre l'influence du choix des valeurs des u_i.

4.2.3 Courbes paramétriques de degré élevé

Il est possible de définir des courbes paramétriques de degré 4, 5 ou plus.

Cela se justifie si l'on dispose des conditions permettant d'attribuer des valeurs pertinentes aux kn coefficients de la matrice A, par exemple, de passer par un grand nombre de points (voir 4.2.4), de respecter des critères aérodynamiques, de résistance de matériau, de longueur minimale de courbe ou d'esthétique, etc..

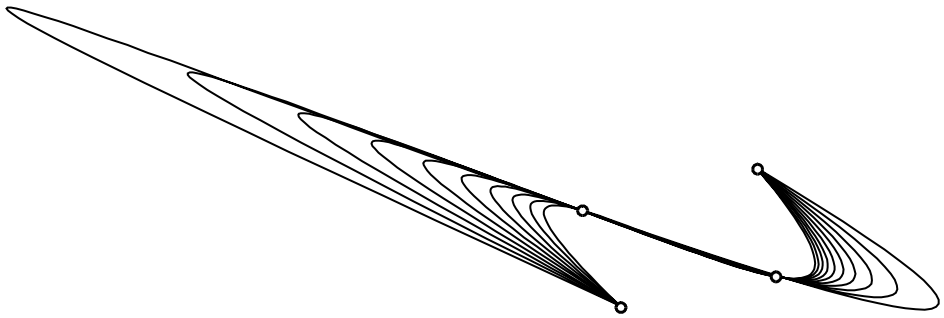


Figure 4.6: Courbes paramétriques cubiques passant par p_0, p_1, p_2 et p_3. Avec $u_0 = 0$, $0.3 \leq u_1 \leq 0.6$, $u_2 = 0.7$ et $u_3 = 1$.

4.2.4 Polynôme de Lagrange

Ainsi la méthode du paragraphe 4.2.2 est généralisable à un nombre $m+1$ de points, elle nécessite la résolution d'un système de $m+1$ équations (il est possible de précalculer l'inverse de la matrice une fois pour toute si l'on choisit des $u_i = i/m \quad i \in [0, m]$). Signalons que c'est en fait une version paramétrique du **polynôme d'interpolation de Lagrange**, pour chaque composante $x, y, z \ldots$ on a

$$P_x(u) = \sum_{i=0}^{m} \frac{(u-u_0)(u-u_1)\cdots(u-u_{i-1})(u-u_{i+1})\cdots(u-u_m)}{(u_i-u_0)(u_i-u_1)\cdots(u-u_{i-1})(u-u_{i+1})\cdots(u_i-u_m)} p_{x_i}.$$

Ceux-ci sont bien ceux que nous cherchons; rappelons, en effet, que le polynôme de Lagrange est le *seul* polynôme de degré m qui passe par $m+1$ points. Nos polynômes paramétriques étant eux aussi de degré m passant par ces points ... ce sont bien les mêmes polynômes.

Cette solution (fig 4.7) n'est généralement pas satisfaisante car, pour des degrés élevés, apparaissent rapidement des **phénomènes d'oscillation** qui ne peuvent être maîtrisés que par un choix judicieux (et difficile) de la répartition des u_i [Mortenson 1985].

4.3 Suite de courbes paramétriques

Nous venons de voir comment définir une courbe paramétrique cubique passant par deux ou quatre points, respectant certaines conditions sur les tangentes, les

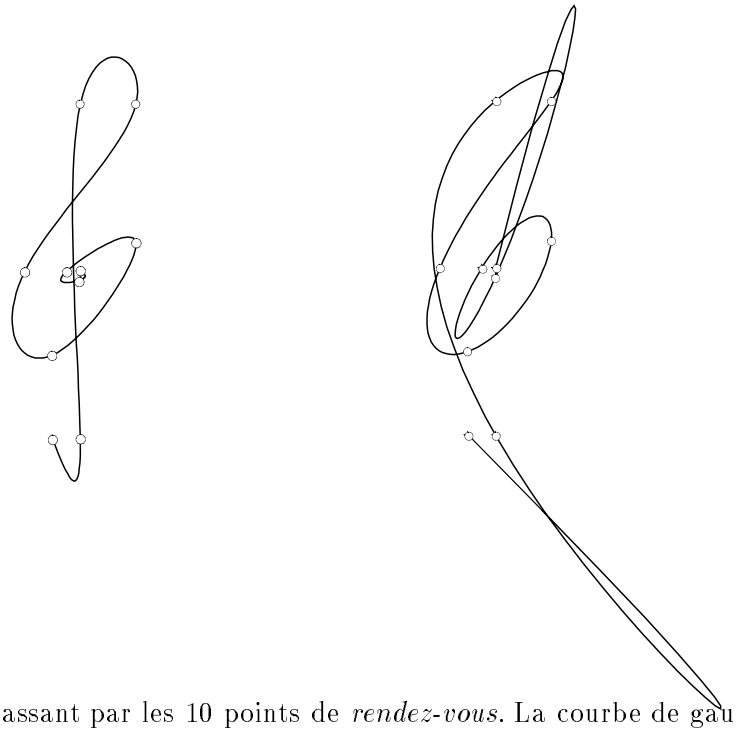


Figure 4.7: Courbes passant par les 10 points de *rendez-vous*. La courbe de gauche est une interpolation par splines cubiques naturelles, celle de droite par des polynômes de Lagrange de degré 9, dans ce cas, on est loin d'obtenir le résultat souhaité.

dérivées secondes, etc..

Le problème que nous nous posons maintenant est le suivant: soient $m+1$ points $p_0, p_1, \ldots, p_m$, comment définir une courbe continue, lisse (continuité des tangentes), qui passe par ces $m+1$ points?

Nous définirons notre courbe d'interpolation comme une **suite de courbes paramétriques cubiques** (que nous noterons $P_i(u)$) sur chacun des intervalles p_{i-1}, p_i.

4.3.1 Conditions à respecter

Pour avoir une courbe *continue* il faut pour tout $i \in [1, m]$

$$\begin{aligned} P_i(0) &= p_{i-1} \\ P_i(1) &= p_i, \end{aligned}$$

pour une courbe *lisse*, pour tout $i \in [1, m-1]$

$$\frac{dP_i}{du}(1) = \frac{dP_{i+1}}{du}(0).$$

Les conditions sur les dérivées indiquent que les directions et les modules des tangentes doivent être les mêmes à gauche et à droite de chacun des points p_i.

En faisant le compte du nombre d'équations et d'inconnues en se plaçant dans ^{n} en supposant les polynômes de degré k, on constate qu'il y a $(3m-1)n$ équations pour $(k+1)mn$ paramètres. Si $k = 3$, ce qui est notre cas, il manque $m+1$ équations.

Cela est dû, d'une part, à ce que les directions et les modules des tangentes ne sont pas précisés, et, d'autre part, à ce que rien n'est dit à propos du comportement de la courbe en p_0 et p_1.

4.3.2 Choix des conditions supplémentaires

Il n'existe, à priori, pas de solution idéale pour déterminer ces conditions, citons, entre autres, celles relatives au comportement des tangentes:

- tangente à la parabole (courbe paramétrique de degré 2) passant par p_{i-1}, p_i, p_{i+1} [Bartels *et al.*, 1987]: la tangente est contrainte à être dans le plan p_{i-1}, p_i, p_{i+1}.
- tangente obtenue par combinaison linéaire des vecteurs p_{i-1}, p_i et p_{i+1} [Kochanek et Bartels, 1984] ou mieux de $p_{i-k}, \ldots p_i, \ldots p_{i+k}$ [Bezier 1987].

Nous choisirons des conditions sur les dérivés secondes, définissant ce qu'on appelle des **splines naturelles**. Celles-ci sont telles que la courbe soit de classe $\mathcal{C}^2$, c'est-à-dire

$$\frac{d^2P_i}{du^2}(1) = \frac{d^2P_{i+1}}{du^2}(0).$$

Nous introduisons ainsi $m-1$ nouvelles conditions, il en manque encore 2, nous en discuterons ci-dessous.

4.3.3 Splines cubiques naturelles

Expression des conditions

Notons A_i ($i \in [1, m]$) la matrice des coefficients de la courbe paramétrique P_i.

Notons, pour $u = 0$

$$U_0 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \qquad U_0' = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \qquad U_0'' = \begin{pmatrix} 0 \\ 2 \\ 0 \\ 0 \end{pmatrix}$$

et $u = 1$

$$U_1 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \qquad U_1' = \begin{pmatrix} 3 \\ 2 \\ 1 \\ 0 \end{pmatrix} \qquad U_1'' = \begin{pmatrix} 6 \\ 2 \\ 0 \\ 0 \end{pmatrix}.$$

Rappelons que, pour $n = 3$, p_i est le vecteur $\begin{pmatrix} p_{i_x} \\ p_{i_y} \\ p_{i_z} \end{pmatrix}$, notons $0 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$.

En posant

$$Q = \begin{pmatrix} U_0 & U_1 & U_1' & U_1'' & & & & & & \\ & & -U_0' & -U_0'' & U_0 & U_1 & \dots & & & \\ & & & & & & \ddots & & & \\ & & & & & & & U_1' & U_1'' & \\ & & & & & & & -U_0' & -U_0'' & U_0 & U_1 \end{pmatrix},$$

les conditions s'expriment alors sous forme matricielle par

$$\begin{bmatrix} A_1 & A_2 & \dots & A_m \end{bmatrix} Q = \begin{bmatrix} p_0 & p_1 & 0 & 0 & p_1 & p_2 & 0 & 0 & \dots & p_{m-1} & p_m \end{bmatrix}.$$

Conditions aux extrémités

Les dimensions des différentes matrices de cette équation sont

$$n \times (k+1)(m), \quad (k+1)(m) \times (4m+2), \quad n \times (4m-2),$$

soit pour $n = 3$ et $k = 3$

$$3 \times 4m, \quad 4m \times (4m+2), \quad 3 \times (4m-2).$$

La matrice n'est pas carrée, il manque deux conditions sur les tangentes en p_0 et p_m. On prendra par exemple

$$\frac{d^2 P_0}{du^2}(0) = 0 \qquad \frac{d^2 P_m}{du^2}(1) = 0$$

ce qui s'écrit

$$A_0 U_0'' = 0 \qquad A_m U_1'' = 0,$$

ou, si la courbe se referme sur elle même,

$$\frac{dP_0}{du}(0) = \frac{dP_m}{du}(1) \qquad \frac{d^2P_0}{du^2}(1) = \frac{d^2P_m}{du^2}(1)$$

ce qui s'écrit

$$A_0 U_0' - A_m U_1' = 0 \qquad A_0 U_0'' - A_m U_1'' = 0.$$

D'autres conditions peuvent être choisies, par exemple celle consistant à rendre la courbe de classe C^3 aux extrémités [Forsythe *et al.*, 1977].

Résolution du système

La matrice Q ainsi complétée est inversible et permet le calcul des A_i (voir programmes en annexes).

Si m est grand l'inversion de Q peut nécessiter beaucoup de temps de calcul. Il est préférable de tenir compte du fait que **Q** est une matrice bande [Numerical Recipes]. Remarquons d'autre part que cette inversion doit être faite une seule fois pour une valeur de m donnée. Un calcul direct de Q^{-1} doit être envisageable, peut-on, par exemple, exprimer Q_{m+1}^{-1} en fonction de Q_m^{-1}?

4.3.4 Approximation des splines naturelles

Afin d'éviter le calcul de Q^{-1}, nous nous proposons non pas de calculer *exactement* ces splines, mais d'en faire une approximation.

Principe

Nous présentonss une méthode consistant à calculer d'abord les tangentes D_i en chacun des p_i. Celles-ci pourraient être obtenues, de manière exacte par résolution d'un grand système d'équations, en imposant certaines conditions aux extrémités, et en les calculant une à une à partir de là [Mortenson 1985, Bartels *et al.*, 1987].

Il semble évident que *tous* les points ne sont pas nécessaires pour calculer chaque tangente D_i, en particulier, le choix des conditions aux extrémités ne devrait

pas influer sur l'allure du milieu de la courbe. Nous proposons une méthode consistant à approximer ces tangentes avec une précision voulue.

Calcul des tangentes

Le système d'équations défini ci-dessus montre que chaque portion P_i de la courbe P dépend de tous les points p_i.

Nous allons d'abord montrer que l'influence des points p_{i+k} sur la forme de la courbe P_i est en raison inverse de l'exponentielle de $|k|$.

Relations entre points et dérivés

Pour cela, exprimons les dérivées D_i en chaque point en fonction des p_j.

Pour $1 \leq i \leq m-1$, on a, d'après les conditions définies en 4.3.1, en combinant les équations de manière à éliminer tous les termes $a_i, b_i, \ldots, d_{i+1}$:

$$D_{i-1} + 4D_i + D_{i+1} = 3(p_{i+1} - p_{i-1})$$

Les conditions pour D_0 et D_m dépendent des choix cités en 4.3.3.

Nous allons d'abord résoudre le système infini, en notant $Z_j = 3(p_{j+1} - p_{j-1})$,

$$\left\{ D_{j-1} + 4D_j + D_{j+1} = Z_j \qquad \forall j \in]-\infty, +\infty[\, . \right.$$

ou, en représentant les équations autour de i,

$$\begin{array}{rcl}
\ldots & & \\
D_{i-2} + 4\,D_{i-1} + D_i = Z_{i-1} & & (i-1) \\
D_{i-1} + 4\,D_i + D_{i+1} = Z_i & & (\ i\ ) \\
D_i + 4\,D_{i+1} + D_{i+2} = Z_{i+1} & & (i+1) \\
D_{i+1} + 4\,D_{i+2} + D_{i+3} = Z_{i+2} & & (i+2) \\
D_{i+2} + 4\,D_{i+3} + D_{i+4} = Z_{i+3} & & (i+3) \\
\ldots & &
\end{array}$$

Par combinaison linéaire de toutes ces équations $(i+k)$ affectées chacune d'un coefficient c_k on obtient

$$
\begin{aligned}
&\cdots \\
&\quad +(c_{-3}+4c_{-2}+c_{-1})D_{i-2} \\
&\qquad +(c_{-2}+4c_{-1}+c_{-0})D_{i-1} \\
&\qquad\quad +(c_{-1}+4c_{0}+c_{+1})D_{i} \\
&\qquad\qquad +(c_{+0}+4c_{+1}+c_{+2})D_{i+1} \\
&\qquad\qquad\quad +(c_{+1}+4c_{+2}+c_{+3})D_{i+2} \\
&\qquad\qquad\qquad +\cdots \\
&\qquad\qquad\qquad\quad = \textstyle\sum_{k=-\infty}^{\infty} c_k Z_{i+k}
\end{aligned}
$$

(nous verrons plus loin que cette somme infinie a effectivement un sens).

Calcul des c_k

Normalisons les coefficients c_k en posant $c_{-1}+4c_0+c_{+1}=1$, et si on choisit, par symétrie, $c_{-k}=c_{+k}$, on a

$$c_{-1}=c_{+1}=1/2-2c_0$$

Si l'on choisit les coefficients de manière que $c_{k-1}+4c_k+c_{k+1}=0$ pour tout $k\neq 0$, le terme en D_{i+k} disparaît et on obtient une expression de D_i indépendante de D_{i+k}.

Il suffit de prendre c_k tels que

- $c_{-k}=c_{+k}\ \forall k\in]-\infty,+\infty[$
- $c_k=-4c_{k+1}-c_{k+2}\ \forall k\in[0,+\infty[$
- $c_{-1}=c_1=1/2-2c_0$
- c_0 tel que la suite c_k tende vers 0 suffisamment rapidement pour que la série infinie converge.

Pour cela il faut résoudre la série formelle

$$\begin{cases} f(n)=-4f(n+1)-f(n+2) \\ f(1)=1/2-2f(0) \\ f(0)=c_0 \end{cases}$$

Ceci se fait de la manière suivante:

Posons

$$f = \sum_{n=0}^{+\infty} u^n f(n),$$

on a alors

$$\sum_{n=0}^{+\infty} u^n f(n) = u^0 f(0) + \sum_{n=0}^{+\infty} u^{n+1} f(n+1),$$

d'où, puisque $f(0) = c_0$

$$\sum_{n=0}^{+\infty} u^{n+1} f(n+1) = f - c_0,$$

de même, avec $f(1) = 1/2 - 2c_0$,

$$\sum_{n=0}^{+\infty} u^{n+2} f(n+2) = f - c_0 - u(1/2 - 2c_0).$$

Or, puisque $f(n) = -4f(n+1) - f(n+2)$, on a

$$\begin{aligned}
\sum_{n=0}^{+\infty} u^n f(n) &= \sum_{n=0}^{+\infty} u^n(-4f(n+1) - f(n+2)) \\
&= -4\sum_{n=0}^{+\infty} u^n f(n+1) - \sum_{n=0}^{+\infty} u^n f(n+2) \\
&= -4\frac{1}{u}\sum_{n=0}^{+\infty} u^{n+1} f(n+1) - \frac{1}{u^2}\sum_{n=0}^{+\infty} u^{n+2} f(n+2)
\end{aligned}$$

c'est-à-dire

$$f = -4\frac{1}{u}(f - c_0) - \frac{1}{u^2}(f - c_0 - u(1/2 - 2c_0))$$

d'où l'on tire finalement

$$f = \frac{c_0 + u(2c_0 + 1/2)}{u^2 + 4u + 1}.$$

f est appelée la *fonction génératrice* de la série formelle, elle peut se mettre sous la forme d'une somme de fractions rationnelles du premier degré:

$$f = \frac{A}{1 - \alpha u} + \frac{B}{1 - \beta u}$$

avec

$$\alpha = \sqrt{3} - 2 \qquad et \qquad \beta = -\sqrt{3} - 2$$

solutions du trinôme $u^2 + 4u + 1$,

et

$$A = \frac{1/2 + c_0(\beta + 2)}{\alpha - \beta}, \qquad B = \frac{1/2 + c_0(\beta + 2)}{\beta - \alpha}.$$

D'où

$$f(n) = A\alpha^n + B\beta^n.$$

On a $\alpha < 1$ et $\beta > 1$. La suite décroît exponentiellement vers 0 si l'on prend c_0 tel que $B = 0$, c'est-à-dire si

$$c_0 = \frac{-1/2}{\beta + 2} = \frac{\sqrt{3}}{6}.$$

On a finalement la solution de la série formelle

$$f(n) = \frac{\sqrt{3}(\sqrt{3} - 2)^n}{6}.$$

Résolution du système

...si l'on choisit $c_k = f(|k|)$ pour tout $k \in]-\infty, +\infty[$, tous les termes autres que D_i s'annulent et on obtient

$$D_i = \sum_{k=-\infty}^{+\infty} c_k Z_{i+k}.$$

Les premières valeurs de c_k sont

k	0	1	2	3	4	5	6	$\cdots$
c_k	0.288	-0.077	0.020	-0.005	0.001	-0.0004	0.0001	$\cdots$

On peut raisonablement supposer que les $Z_{i+k} = 3(p_{i+k+1} - p_{i+k-1})$ ne croissent pas exponentiellement et donc que la série infinie converge, puisque c_k tend vers 0 exponentiellement.

D'autre part, puisque $c_k = c_{-k}$, on a pour tout $k \geq 0$

$$\begin{aligned} & c_k(p_{i+k+1} - p_{i+k-1}) + c_{-k}(p_{i-k+1} - p_{i-k-1}) \\ = \;& c_k[(p_{i+(k+1)} - p_{i-(k+1)}) - (p_{i+(k-1)} - p_{i-(k-1)})]. \end{aligned}$$

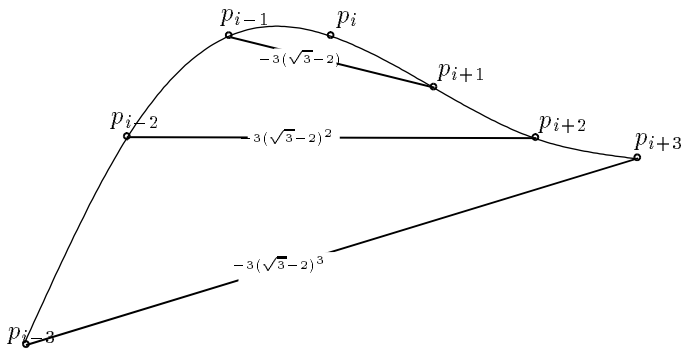


Figure 4.8: La tangente en p_i est obtenue par combinaison linéaire des vecteurs $p_{i+k}p_{i-k}$ affectés des coefficients $-3(\sqrt{3}-2)^k$.

Chaque terme $(p_{i+k} - p_{i-k})$ apparaît deux fois, affecté des coefficients c_{k-1} et $-c_{k+1}$, on a $c_{0-1} - c_{0+1} = 0$ et, pour $k > 0$

$$\begin{aligned} c_{k-1} - c_{k+1} &= \frac{\sqrt{3}(\sqrt{3}-2)^k}{6}\left(\frac{1}{\sqrt{3}-2} - \frac{\sqrt{3}-2}{1}\right) \\ &= -(\sqrt{3}-2)^k \end{aligned}$$

Finalement

$$D_i = 3\sum_{k=1}^{+\infty} -(\sqrt{3}-2)^k(p_{i+k} - p_{i-k})$$

En première approximation D_i est parallèle à $p_{i+1} - p_{i-1}$, puis viennent les termes correctifs dus à $p_{i+2} - p_{i-2}$, $p_{i+3} - p_{i-3}$, etc., la figure 4.8 illustre la construction de la tangente D_i.

4.3.5 Construction géométrique de la courbe

Nous présentons maintenant la méthode de construction de la courbe d'interpolation qui consiste à *tendre des ficelles* entre les points d'ancrage que sont les points p_i et des points l_i et r_i situés sur les tangentes de part et d'autre des p_i.

La courbe obtenue est encore la même, son mode de construction traduit certaines propriétés géométriques que nous utiliserons par la suite en particulier pour l'interpolation des quaternions au chapitre 5. Cette méthode consiste à calculer d'abord les tangentes en chaque p_i, puis les points de la courbe.

Soit r_{i-1} le point situé "à droite"[1] sur la tangente en p_{i-1} du coté p_i et l_i le point situé "à gauche"[1] sur la tangente en p_i du coté p_{i-1}.

Tendons des ficelles entre p_{i-1} et r_{i-1}, entre r_{i-1} et l_i, et entre l_i et p_i.

Sur chacun de ces segments nous définissons, relativement au paramètre u, les points $d_i(u)$, $m_i(u)$ et $g_i(u)$ tels que

$$\begin{aligned} d_i(u) &= (1-u)p_{i-1} + ur_{i-1} \\ m_i(u) &= (1-u)r_{i-1} + ul_i \\ g_i(u) &= (1-u)l_i + up_i \end{aligned}$$

Nous choisirons à nouveau des points $a_i(u)$ et $b_i(u)$ sur les deux segments d_im_i et m_ig_i

$$\begin{aligned} a_i(u) &= (1-u)d_i(u) + um_i(u) \\ b_i(u) &= (1-u)m_i(u) + ug_i(u) \end{aligned}$$

et enfin $P_i(u)$ sur cette *corde* tendue entre a_i et b_i par

$$P_i(u) = (1-u)a_i(u) + ub_i(u).$$

Lorsque u varie de 0 à 1, $P_i(u)$ décrit la courbe de p_{i-1} à p_i.

En développant P_i par rapport à p_{i-1}, r_{i-1}, l_i et p_i on obtient

$$P_i = (1-u)^3p_{i-1} + 3(1-u)^2ur_{i-1} + 3u^2(1-u)l_i + u^3p_i,$$

polynômes de Bernstein de degré 3.

En identifiant les coefficients de ces polynômes avec ceux définis précédemment on constate que pour obtenir la même équation il suffit de prendre

$$r_{i-1} = p_{i-1} + \frac{1}{3}\frac{dp_i}{du}(0) \qquad\qquad l_i = p_i - \frac{1}{3}\frac{dp_i}{du}(1).$$

[1] les termes *à droite* et *à gauche* n'ont ici aucune signification géométrique, ils ne servent que de moyens mnémotechniques pour indiquer que **l** (comme **l***eft*) est situé avant **r** (comme **r***ight*), ... si les points sont ordonnés de gauche à droite.

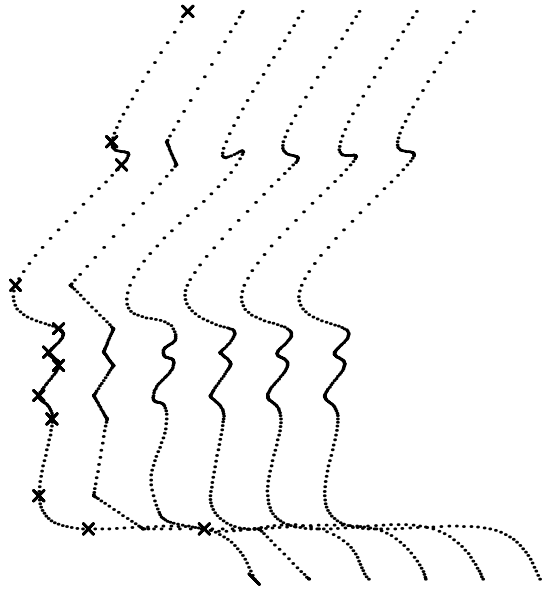


Figure 4.9: Interpolation par approximation de splines naturelles cubiques. La première courbe (avec points de rendez-vous (extraits de [Piegl 1987])) et la dernière correspondent à un calcul exact, les autres à une approximation tenant compte respectivement d'un nombre K de termes égal à 0, 1, 2 et 3.

Résumons

Voilà, nous venons de présenter une méthode simple et efficace de calcul de la courbe d'interpolation par splines cubiques naturelles qui ne demande pas la résolution d'un grand système d'équations. L'approximation est excellente, la précision est plus ou moins grande suivant le nombre de termes mis en jeu et, sauf cas exceptionnels, trois termes suffisent. Sur la figure 4.9 est représentée l'interpolation par splines naturelles cubiques du *profil* extrait de l'article [Piegl 1987].

4.3.6 Réalisation pratique

Soient $p_0, p_1, \cdots, p_m$ les $m+1$ points *rendez-vous*. Soient N le nombre total de points à déterminer, nous supposerons dans un premier temps que la répartition des points en fonction du temps est uniforme (voir en détail 4.4). Soit K le nombre de termes à prendre en considération pour l'approximation des dérivées.

Il faudra calculer N/m points pour chacune des m portions de courbe P_i.

Pour tout intervalle i de p_{i-1} a p_i $(i \in [1, m])$

$r_{i-1} = -3 \sum_{k=0}^{K} (\sqrt{3} - 2)^k (p_{i-1+k} - p_{i-1-k})$

$l_i = -3 \sum_{k=0}^{K} (\sqrt{3} - 2)^k (p_{i+k} - p_{i-k})$

pour u de 0 a 1 par increment de N/m

calcul de $d(u)$, $m(u)$, $g(u)$ en fonction de p_{i-1}, p_i, r_{i-1} et l_i

calcul de $a(u)$, $b(u)$ en fonction de $d(u)$, $m(u)$, $g(u)$

calcul de $P_i(u)$ en fonction de $a(u)$ et $b(u)$.

4.4 Interpolation en fonction du temps

4.4.1 Présentation du problème

Nous allons maintenant introduire la notion de **temps**, c'est-à-dire que nous affecterons une **date** à chacun des points de la courbe. Jusqu'à maintenant, nous avions implicitement supposé que les points de *rendez-vous* p_i étaient répartis régulièrement en fonction du temps, c'est-à-dire que la durée séparant les instants t_{i-1}, date de présence en p_{i-1}, et t_i, date de présence en p_i, était constante quel que soit i.

La forme de la courbe ainsi que la vitesse de parcours dépendent grandement de la durée relative de ces intervalles. Pour obtenir un résultat correct, il est nécessaire d'adapter tous les calculs présentés précédemment.

4.4.2 Nouvelles conditions à respecter

Nous exprimons maintenant la courbe $\mathbf{P}$ en fonction d'une variable t représentant le temps.

Nous écrirons, pour $t \in [t_{i-1}, t_i]$,

$$P(t) = a_i \left(\frac{t - t_{i-1}}{t_i - t_{i-1}} \right)^3 + b_i \left(\frac{t - t_{i-1}}{t_i - t_{i-1}} \right)^2 + c_i \left(\frac{t - t_{i-1}}{t_i - t_{i-1}} \right) + d_i.$$

Les conditions s'expriment maintenant par

- passage de la courbe aux points de *rendez-vous*

$$P(t_i) = p_i$$

qui se traduit par

$$a_i + b_i + c_i + d_i = d_{i+1} = p_i$$

- dérivés premières continues

$$\frac{dP(t)}{dt}\Big|_{t=t_i^-} = \frac{dP(t)}{dt}\Big|_{t=t_i^+}$$

c'est-à-dire, avec $u = \frac{t - t_{i-1}}{t_i - t_{i-1}}$ et $v = \frac{t - t_i}{t_{i+1} - t_i}$

$$\frac{1}{(t_i - t_{i-1})}\frac{dP_i(u)}{du} = \frac{1}{(t_{i+1} - t_i)}\frac{dP_{i+1}(v)}{dv}$$

- dérivés secondes continues

$$\frac{d^2P(t)}{dt^2}\Big|_{t=t_i^-} = \frac{d^2P(t)}{dt^2}\Big|_{t=t_i^+}$$

c'est-à-dire

$$\frac{1}{(t_i - t_{i-1})^2}\frac{d^2P_i(u)}{du^2}\Big|_{u=1} = \frac{1}{(t_{i+1} - t_i)^2}\frac{d^2P_{i+1}(v)}{dv^2}\Big|_{v=0}.$$

Les équations du paragraphe 4.3.3 sont de ce fait à reconsidérer, nous devons maintenant écrire

$$U_0 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \qquad U_0' = \frac{1}{(t_{i+1} - t_i)}\begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \qquad U_0'' = \frac{1}{(t_{i+1} - t_i)^2}\begin{pmatrix} 0 \\ 2 \\ 0 \\ 0 \end{pmatrix}$$

et $u = 1$

$$U_1 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \qquad U_1' = \frac{1}{(t_{i+1} - t_i)}\begin{pmatrix} 3 \\ 2 \\ 1 \\ 0 \end{pmatrix} \qquad U_1'' = \frac{1}{(t_{i+1} - t_i)^2}\begin{pmatrix} 6 \\ 2 \\ 0 \\ 0 \end{pmatrix}.$$

Ceci revient à diviser les colonnes $4(i-1)+3$ de la matrice $\mathbf{Q}$ par l'intervalle de temps $(t_i - t_{i-1})$ et à diviser les colonnes $4(i-1)+4$ par $(t_i - t_{i-1})^2$.

4.4.3 Résolution du système

La nouvelle matrice $\mathbf{Q}$ ainsi obtenue, qui est toujours une matrice bande, est inversible si tous les intervalles de temps sont non nuls. En annexe figurent les programmes, on y trouvera les différents cas de conditions aux extrémités.

4.4.4 L'approximation est-elle encore possible?

Au paragraphe 4.3.4 nous avions présenté une méthode d'approximation des splines naturelles cubiques en calculant une approximation des dérivés en chacun des points p_i. Pour appliquer la même méthode ici nous devons établir le système d'équations permettant de définir ces dérivés D_i. Par une combinaison linéaire adéquate de toutes les équations, relatives aux conditions à respecter, en fonction des a_i, b_i, c_i, d_i, ainsi que celles définissant les p_i et D_i, on obtient les équations $\forall j \in]-\infty, +\infty[$

$$\frac{D_{j-1}}{t_j - t_{j-1}} + \frac{2D_j}{t_j - t_{j-1}} + \frac{2D_j}{t_{j+1} - t_j} + \frac{D_{j+1}}{t_{j+1} - t_j} = 3\frac{p_j - p_{j-1}}{(t_j - t_{j-1})^2} + 3\frac{p_{j+1} - p_j}{(t_{j+1} - t_j)^2}$$

On retrouve bien l'équation de 4.3.4 lorsque $t_i - t_{i-1} = 1 \qquad \forall i$.

Par combinaison linéaire de toutes ces équations affectées d'un coefficient h_j, on obtient les D_i affectés d'un terme K_i

$$K_i = \frac{h_{i-1}}{t_i - t_{i-1}} + \frac{2h_i}{t_i - t_{i-1}} + \frac{2h_i}{t_{i+1} - t_i} + \frac{h_{i+1}}{t_{i+1} - t_i}$$

Il ne reste plus qu'à choisir les h_j pour que

- $K_i = 1$
- $K_j = 0 \qquad \forall j \neq i$
- la somme infinie associée au second membre de l'équation converge.

Une solution sous la forme d'une expression formelle simple n'est pas possible, il est nécessaire de calculer ces valeurs pour pour chaque suite t_i. Nous savons calculer une valeur approximative des coefficients h en inversant des matrices dont la taille ne dépend que de la précision souhaitée (des matrices de rang 5 sont suffisantes).

4.5 Interpolation dans AGraph

4.5.1 Trajectoire et facteur d'échelle

Ces techniques d'interpolation s'appliquent sans problème au calcul d'une trajectoire dans ^{3}.

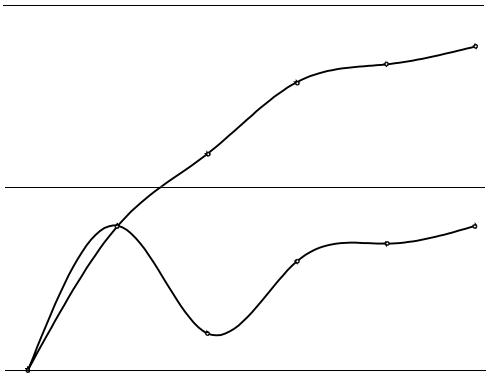


Figure 4.10: Interpolation d'une variable périodique. Les points de *rendez-vous* 3, 4, 5 et 6 peuvent être considérés comme étant situés soit dans la même période que les points 1 et 2, soit dans la période suivante.

Il en est de même pour le facteur d'échelle si l'on autorise des valeurs négatives, dans le cas contraire, il est indispensable de corriger les valeurs obtenues de manière qu'elles restent positives.

Cela peut être fait en remplaçant simplement ces valeurs négatives par 0, mais présente l'inconvénient de ne plus être dérivable. Il y a un choc. Pour éviter ça nous proposons une solution qui consiste à effectuer un changement de variable.

4.5.2 Couleur, saturation, luminosité

Interpolation séparée

La couleur est définie par une valeur comprise entre 0 et 360, la saturation et le contraste entre 0 et 1. On peut interpoler chacune des variables séparement, en tenant compte du fait que les valeurs de couleur se retrouvent modulo 360, qu'il ne faut pas obtenir de valeur de saturation et de contraste en dehors de l'intervalle [fig.4.10, 4.11].

Comment respecter ces contraintes? En ce qui concerne l'interpolation de la couleur, faut-t-il aller directement d'une valeur à l'autre ou faut-il effectuer plusieurs tours (on a d'ailleurs le même problème pour la rotation)? La figure 4.12 présente en trait fin une interpolation directe, en trait gras une interpolation admettant des tours supplémentaires, cette dernière solution est souvent la plus lisse.

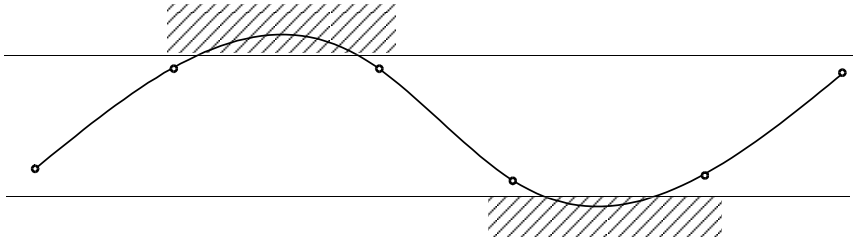


Figure 4.11: Interpolation d'une variable définie dans un intervalle. Sans précautions particulières, certaines parties de la courbe d'interpolation peuvent être en dehors de l'intervalle (parties hachurées).

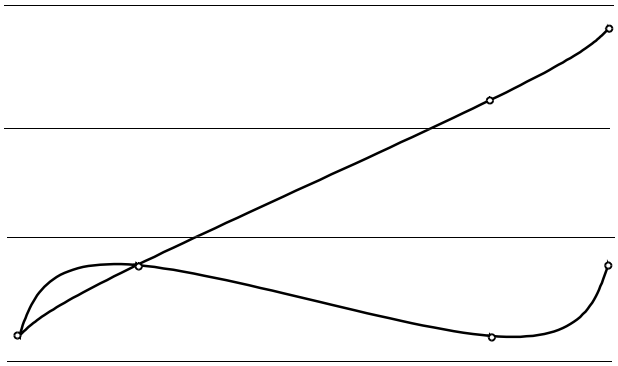


Figure 4.12: Interpolation d'une variable périodique *modulo* un certain nombre de périodes. Ici l'une des courbe oscille entre les valeurs extrèmes, l'autre recouvre trois périodes.

Interpolation globale

On peut également interpoler conjointement les éléments du couple couleur-saturation. La courbe obtenue devant être entièrement située dans le disque unité.

4.5.3 Orientations

Nous adaptons maintenant ces méthodes d'interpolation au calcul des orientations en représentant celles-ci par des quaternions.

Tel est le but du chapitre suivant.

Chapitre 5

Rotations et quaternions

La trajectoire tridimensionnelle, la taille et les paramètres de couleur et de luminosité des objets rigides constituant les acteurs (cf. chapitre 2) de nos séquences d'animation sont calculés par les méthodes d'interpolation décrites au chapitre précédent.

Il reste à définir les orientations dans l'espace que doivent adopter ces acteurs. Pour cela nous associerons, à chaque instant, une rotation autour du point de l'objet décrivant la trajectoire. Ces rotations, calculées par interpolation, devront, comme pour la trajectoire, donner l'illusion d'un mouvement continu et lisse.

5.1 Rotations

Un déplacement dans ^{3} peut être décomposé en une translation et une **rotation autour de l'origine** [Euler 1758].

La position d'un objet rigide indéformable peut ainsi être définie par les coordonnées d'un de ses points et par une orientation (rotation) de l'objet autour de ce point; tout mouvement dans l'espace ^{3} peut être obtenu par une suite de translations et d'orientations.

5.1.1 Comment définir une rotation?

Une rotation peut être définie de plusieurs façons

- par les **angles d'Euler** (rotation autour de trois axes orthogonaux mobiles)

- par des rotations autour d'axes orthogonaux fixes
- par la donnée d'un **axe** de rotation et d'un **angle** autour de cet axe
- par une **matrice** 3×3
- par un **quaternion** de norme 1
- etc.

Quelle que soit la définition utilisée, trois paramètres indépendants sont nécessaires et suffisants.

Angles d'Euler

La rotation est définie par le produit de trois rotations autour d'axes orthogonaux [Euler 1758]. Il existe ainsi beaucoup de manières de définir la rotation, autant qu'il y a de manière de choisir trois axes parmi Ox, Oy et Oz.

Généralement on choisit une rotation autour de Oz qui fait tourner le plan xOy d'un angle γ, puis une rotation d'axe Oy_1 (Oy après rotation autour de Oz), qui fait tourner x_1Oz d'un angle β, et enfin une troisième d'axe Oz_2, qui fait tourner le plan x_2Oy_2 d'un angle α (attention à l'ordre).

La matrice correspondante est obtenue par le produit

$$R = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix} \cdot \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

soit

$$\begin{pmatrix} \cos\alpha\cos\beta\cos\gamma - \sin\alpha\sin\gamma & -\cos\alpha\cos\beta\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\sin\beta \\ \sin\alpha\cos\beta\cos\gamma + \cos\alpha\sin\gamma & -\sin\alpha\cos\beta\sin\gamma + \cos\alpha\cos\gamma & -\sin\alpha\sin\beta \\ -\sin\beta\cos\gamma & \sin\beta\cos\gamma & \cos\beta \end{pmatrix}$$

Réciproquement, connaissant la matrice R on peut calculer β par r_{33}, puis γ par r_{31} et r_{32} ainsi que α par r_{13} et r_{23}. Notons que le choix du signe de β est arbitraire et qu'il détermine les valeurs de α et γ.

Les angles d'Euler, bien que souvent employés, ne sont pas d'une manipulation aisée, certain les considérant même comme une *atrocité mathématique* [de Casteljau 1987,

p 57]. La composition de rotations ne se fait pas simplement, il ne suffit pas d'additionner α_1 à α_2 etc., les axes étant également soumis aux rotations. Le choix des axes et l'ordre des rotations doivent être respectés, les conventions sont multiples (zxz pour certains physiciens, zyz en aéronautique, xyz pour d'autres ...).

Ils sont par contre indispensables lorque les axes de rotations doivent être liés à l'objet, mécanique céleste, aéronautique, gyroscopes etc..

Axes fixes orthogonaux

Dans la définition des angles d'Euler, les axes Oy_1 et Oz_2 sont obtenus par rotation des axes Oy et Oz du repère. Il est possible de définir des rotations autour des axes fixes Ox, Oy et Oz. Pour calculer la matrice de rotation associée **il suffit** d'effectuer le produit de matrices dans l'**ordre inverse** de celui défini pour les angles d'Euler.

Ainsi si l'on applique une rotation R_z autour de Oz puis R_y autour de Oy la matrice résultante est

$$R_{yz} = R_z R_y$$

et non pas $R_y R_z$ comme pour les angles d'Euler. Nous montrerons facilement à l'aide des quaternions en 5.3.4 que si Oy_1 est l'axe obtenu par la rotation de Oy par R_z alors

$$R_z R_y = R_{y_1} R_z.$$

Les angles de rotation de R_y et R_{y_1} sont les mêmes.

D'une manière générale en notant R_i la rotation autour d'un axe quelconque A_i, le produit

$$R_k R_{k-1} \cdots R_1 R_0$$

représente soit la composition des rotations R_0 puis R_1 puis R_2 etc. autour des axes A_i' obtenus par les rotations successives $R_0 R_1 \cdots R_{i-1}$ des axes A_i soit la composition R_k puis R_{k-1} puis R_{k-2} autour des axes fixes A_i.

Cette propriété du produit de matrice est importante et est utilisée pour la programmation des boutons de manipulation des logiciels de graphisme interactif. Les deux types de rotations sont généralement nécessaires, les axes étant liés soit à l'écran soit à l'objet graphique que l'on manipule.

Axe et angle

Soit $A = \begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix}$ avec $\|A\| = 1$ l'axe de la rotation et θ l'angle. La matrice de rotation est

$$\begin{pmatrix} a_x a_x(1-\cos\theta)+\cos\theta & a_y a_x(1-\cos\theta)-a_z\sin\theta & a_z a_x(1-\cos\theta)+a_y\sin\theta \\ a_x a_y(1-\cos\theta)+a_z\sin\theta & a_y a_y(1-\cos\theta)+\cos\theta & a_z a_y(1-\cos\theta)-a_x\sin\theta \\ a_x a_z(1-\cos\theta)-a_y\sin\theta & a_y a_z(1-\cos\theta)+a_x\sin\theta & a_z a_z(1-\cos\theta)+\cos\theta \end{pmatrix}$$

Réciproquement, de cette matrice R on extrait facilement θ par

$$Trace(R) = 1 + 2\cos\theta$$

puis les coefficients directeurs de l'axe A [Altmann 1986].

Cette méthode de représentation est principalement utilisée lorsqu'il s'agit d'exprimer une rotation autour d'un axe quelconque autre que ceux du repère ou ceux de l'objet.

Matrices de rotations

Les matrices 3×3 facilitent évidemment beaucoup les calculs, la composition de rotations s'expriment par un simple produit, la rotation inverse par la transposée.

De plus les matrices de rotation peuvent être composées à d'autres types de matrices, homothétie, inversion, matrices quelconques etc.. Les opérations de l'algèbre des matrices s'appliquent sans restriction, les rotations n'en sont qu'un cas particulier.

5.2 Quaternions

Les quaternions, introduits par Hamilton afin de définir le quotient et le produit de deux vecteurs quelconques dans ^{3} [Hamilton 1866], se sont révélés être une manière élégante d'exprimer une rotation [Shoemake 1985] [Yahia et Gagalowicz, 1989].

Ils sont également une généralisation à quatre dimensions des nombres complexes, c'est ainsi que, dans un premier temps, nous les présentons [Bass].

5.2.1 Définition des quaternions

... Hamilton, qui cherchait à généraliser les nombres complexes en dimension trois, réalisa le matin du lundi 16 octobre 1843, alors qu'il se promenait en compagnie de Lady Hamilton sur les berges du Royal Canal à Dublin non loin de Broome Bridge, que quatre dimensions étaient nécessaires[Altmann 1986, p 12] ...

De même qu'à un point (a, b) du plan on associe le nombre complexe $a + ib$, on associera à un point (t, x, y, z) de ^{4} le quaternion $t + xi + yj + zk$ où $\mathbf{1}$, i, j et k sont les "vecteurs unitaires" des quatres axes de cordonnées Ot, Ox, Oy et Oz.

A cet espace vectoriel des quaternions on donne une stucture d'algèbre en définissant une opération de multiplication associative et distributive par rapport à l'addition définie sur les vecteurs de base par le tableau:

$\times$	1	i	j	k
1	1	i	j	k
i	i	-1	k	$-j$
j	j	$-k$	-1	i
k	k	j	$-i$	-1

Notons que **la multiplication n'est pas commutative**.

5.2.2 Somme, produit et multiplication par un scalaire

Soient $q = t + xi + yj + zk$ et $q' = t' + x'i + y'j + z'k$ deux quaternions et $\lambda \in$.

On aura

$$\begin{aligned} q + q' &= (t + t') + (x + x')i + (y + y')j + (z + z')k \\ qq' &= (tt' - xx' - yy' - zz') \\ &\quad + (tx' + xt' + yz' - zy')i \\ &\quad + (ty' + yt' + zx' - xz')j \\ &\quad + (tz' + zt' + xy' - yx')k \\ \lambda q &= \lambda t + \lambda xi + \lambda yj + \lambda zk. \end{aligned}$$

5.2.3 Conjugué, norme, inverse

On définit le quaternion conjugué

$$\overline{q} = t - xi - yj - zk.$$

On remarquera que

$$\begin{aligned} q + \overline{q} &= 2t \qquad \in , \\ q\overline{q} &= t^2 + x^2 + y^2 + z^2 \qquad \in {}^+ \end{aligned}$$

et

$$\overline{qq'} = \overline{q'}\,\overline{q}.$$

On introduit la norme

$$\|q\| = \sqrt{q\overline{q}}$$

et l'on a

$$\|qq'\| = \|q\|\|q'\|.$$

Enfin, pour tout quaternion $q \neq 0$, on définit le quaternion inverse

$$q^{-1} = \frac{\overline{q}}{\|q\|^2}.$$

Les quaternions munis de l'opération de multiplication et de l'inverse possèdent la stucture de corps, c'est **le seul corps non commutatif de dimension finie** sur le corps des nombres réels [Verley 1988].

Quaternions

5.3 Quaternions et rotations

A un nombre complexe $e^{i\theta}$ du cercle unité peut être associée une rotation dans le plan d'angle θ autour de l'origine ...de même à toute rotation dans ^{3} peut être associé un quaternion de norme 1.

Nous allons maintenant montrer qu'il existe un isomorphisme entre les rotations d'axes concourants passant par l'origine de ^{3} et les quaternions de norme 1.

5.3.1 Le groupe des quaternions de norme 1

Les quaternions de norme 1 munis de la multiplication et de l'inverse forment un groupe.

En effet $\forall q, q'$ tels que $\|q\| = \|q'\| = 1$ on a

$$\|qq'\| = \|q\|\|q'\| = 1$$

ainsi que si $q \neq 0$

$$\|q^{-1}\| = \|\frac{\overline{q}}{\|q\|^2}\| = \frac{\|q\|}{\|q\|^2} = 1.$$

5.3.2 Du quaternion à la rotation

Produit de 3 quaternions $r\xi r'$

Calculons explicitement le produit de trois quaternions r, ξ et r' en factorisant par rapport à ξ.

$$\begin{aligned}
&(a + bi + cj + dk)(t + xi + yj + zk)(a' + b'i + c'j + d'k) \\
&= \Big[(at - bx - cy - dz) \\
&\quad + (ax + bt + cz - dy)i \\
&\quad + (ay + ct + dx - bz)j \\
&\quad + (az + dt + by - cx)k\Big](a' + b'i + c'j + d'k) \\
&= \Big[(at - bx - cy - dz)a' - (ax + bt + cz - dy)b' \\
&\qquad - (ay + ct + dx - bz)c' - (az + dt + by - cx)d'\Big] \\
&\quad + \Big[(at - bx - cy - dz)b' + (ax + bt + cz - dy)a' \\
&\qquad + (ay + ct + dx - bz)d' - (az + dt + by - cx)c'\Big]i \\
&\quad + \Big[(at - bx - cy - dz)c' + (ay + ct + dx - bz)a' \\
&\qquad + (az + dt + by - cx)b' - (ax + bt + cz - dy)d'\Big]j \\
&\quad + \Big[(at - bx - cy - dz)d' + (az + dt + by - cx)a' \\
&\qquad + (ax + bt + cz - dy)c' - (ay + ct + dx - bz)b'\Big]k
\end{aligned}$$

et, après développement et mise en facteur

$$
\begin{aligned}
&(a+bi+cj+dk)(t+xi+yj+zk)(a'+b'i+c'j+d'k)\\
&= \quad \Big[(aa'-bb'-cc'-dd')t+(-ba'-ab'-dc'+cd')x\\
&\qquad +(-ca'+db'-ac'-bd')y+(-da'-cb'+bc'-ad')z\Big]\\
&\quad +\Big[(ab'+ba'+cd'-dc')t+(-bb'+aa'+dd'+cc')x\\
&\qquad +(-cb'-da'+ad'-bc')y+(-db'+ca'-bd'-ac')z\Big]i\\
&\quad +\Big[(ac'+ca'+db'-bd')t+(-bc'+da'-cb'-ad')x\\
&\qquad +(-cc'+aa'+bb'+dd')y+(-dc'-ba'+ab'-cd')z\Big]j\\
&\quad +\Big[(ad'+da'+bc'-cb')t+(-bd'-ca'+ac'-db')x\\
&\qquad +(-cd'+ba'-dc'-ab')y+(-dd'+aa'+cc'+bb')z\Big]k
\end{aligned}
$$

Produit $r\xi\overline{r}$

Remplaçons $(a'+b'i+c'j+d'k)$ par le quaternion conjugué de $(a+bi+cj+dk)$ on obtient

$$
\begin{aligned}
&(a+bi+cj+dk)(t+xi+yj+zk)(a-bi-cj-dk)\\
&= \quad \Big[(a^2+b^2+c^2+d^2)t+(-ba+ab+dc-cd)x\\
&\qquad +(-ca-db+ac+bd)y+(-da+cb-bc+ad)z\Big]\\
&\quad +\Big[(-ab+ba-cd+dc)t+(b^2+a^2-d^2-c^2)x\\
&\qquad +(cb-da-ad+bc)y+(db+ca+bd+ac)z\Big]i\\
&\quad +\Big[(-ac-ca-db+bd)t+(bc+da+cb+ad)x\\
&\qquad +(c^2+a^2-b^2-d^2)y+(dc-ba-ab+cd)z\Big]j\\
&\quad +\Big[(-ad+da-bc+cb)t+(bd-ca-ac+db)x\\
&\qquad +(cd+ba+dc+ab)y+(d^2+a^2-c^2-b^2)z\Big]k
\end{aligned}
$$

avec r normé

$$a^2+b^2+c^2+d^2=1$$

$$(a+bi+cj+dk)(t+xi+yj+zk)(a-bi-cj-dk)$$

$$
\begin{aligned}
= \quad & \Big[(1)t + (0)x + (0)y + (0)z\Big] \\
& + \Big[(0)t + (a^2 + b^2 - c^2 - d^2)x - 2(ad - bc)y + 2(ac + bd)z\Big]i \\
& + \Big[(0)t + 2(ad + bc)x + (a^2 - b^2 + c^2 - d^2)y - 2(ab - cd)z\Big]j \\
& + \Big[(0)t - 2(ac - bd)x + 2(ab + cd)y + (a^2 - c^2 - b^2 + d^2)z\Big]k
\end{aligned}
$$

$r\xi\overline{r}$ peut s'écrire sous une forme matricielle

On voit que la composante réelle $\mathcal{R}(r\xi\overline{r})$ est égale à t. D'autre part la partie imaginaire $\mathcal{I}(r\xi\overline{r})$ peut s'écrire

$$\mathcal{I}(r\xi\overline{r}) = M \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

où M est la matrice 3×3

$$M = \begin{pmatrix} 1 - 2c^2 - 2d^2 & 2bc - 2ad & 2bd + 2ac \\ 2bc + 2ad & 1 - 2d^2 - 2b^2 & 2cd - 2ab \\ 2bd - 2ac & 2cd + 2ab & 1 - 2b^2 - 2c^2 \end{pmatrix}$$

M est une matrice de rotation

Il est facile de vérifier que $\det M = 1$ et que M est une matrice orthogonale, c'est à dire que si $M_{\_i}$ et $M_{\_j}$ sont deux colonnes de M alors le produit scalaire $M_{\_i} \cdot M_{\_j}$ est nul si $i \neq j$ et 1 sinon.

Axe et angle de la rotation

Déterminons l'axe Δ et l'angle θ de cette rotation, pour cela il faut calculer δ_x, δ_y et δ_z les coefficients directeurs de Δ tels que

$$M\Delta = \Delta$$

c'est-à-dire

$$\begin{cases} (1 - 2c^2 - 2d^2)\delta_x + (2bc - 2ad)\delta_y + (2bd + 2ac)\delta_z = \delta_x \\ (2bc + 2ad)\delta_x + (1 - 2d^2 - 2b^2)\delta_y + (2cd - 2ab)\delta_z = \delta_y \\ (2bd - 2ac)\delta_x + (2cd + 2ab)\delta_y + (1 - 2b^2 - 2c^2)\delta_z = \delta_z \end{cases}$$

ou

$$\begin{cases} (-c^2-d^2)\delta_x + (bc-ad)\delta_y + (bd+ac)\delta_z = 0 \\ (bc+ad)\delta_x + (-d^2-b^2)\delta_y + (cd-ab)\delta_z = 0 \\ (bd-ac)\delta_x + (cd+ab)\delta_y + (-b^2-c^2)\delta_z = 0 \end{cases}$$

Supposons $c^2+d^2 \neq 0$ (sinon on fait le même raisonnement avec δ_y ou δ_z).

On a alors

$$\delta_x = \frac{(bc-ad)\delta_y + (bd+ac)\delta_z}{c^2+d^2}$$

qui, reporté dans les équations (2) et (3), donne

$$\begin{cases} ((bc+ad)(bc-ad) + (-d^2-b^2)(c^2+d^2))\delta_y/(c^2+d^2) \\ \quad + ((bc+ad)(bd+ac) + (cd-ab)\delta_z/(c^2+d^2) = 0 \\ ((bd-ac)(bc-ad) + (cd+ab)(c^2+d^2))\delta_y/(c^2+d^2) \\ \quad + ((bd-ac)(bd+ac) + (-b^2-c^2)(c^2+d^2))\delta_z/(c^2+d^2) = 0 \end{cases}$$

c'est-à-dire

$$\begin{cases} (b^2c^2 - a^2d^2 - c^2d^2 - d^2d^2 - b^2c^2 - b^2d^2)\delta_y \\ \quad + (b^2cd + abc^2 + abd^2 + a^2cd + c^2cd + cd^2d - abc^2 - abd^2)\delta_z = 0 \\ (b^2dc - abd^2 - abc^2 + a^2cd + dc^2c + cd^2d + abc^2 + abd^2)\delta_y \\ \quad + (b^2d^2 - a^2c^2 - b^2c^2 - c^2c^2 - b^2d^2 - c^2d^2)\delta_z = 0 \end{cases}$$

et, avec $a^2+b^2+c^2+d^2 = 1$,

$$\begin{cases} -d^2\delta_y + cd\delta_z = 0 \\ cd\delta_y - c^2\delta_z = 0 \end{cases}$$

d'où il vient

$$\begin{cases} \delta_y = \alpha c \quad \alpha \in {}^* \\ \delta_z = \alpha d \end{cases}$$

et par conséquent

$$\begin{aligned} \delta_x &= \frac{(bc-ad)\delta_y + (bd+ac)\delta_z}{c^2+d^2} \\ &= \frac{\alpha(bc^2-acd) + \alpha(bd^2+acd)}{c^2+d^2} \\ &= b \end{aligned}$$

Donc Δ est donné par

$$\Delta = \alpha \begin{pmatrix} b \\ c \\ d \end{pmatrix}$$

L'angle θ est donné par le produit scalaire

$$\cos\theta = \frac{1}{u^2+v^2+w^2} \begin{pmatrix} u \\ v \\ w \end{pmatrix} \cdot M \begin{pmatrix} u \\ v \\ w \end{pmatrix}$$

où $\begin{pmatrix} u \\ v \\ w \end{pmatrix}$ est un vecteur orthogonal à Δ par exemple $\begin{pmatrix} -c \\ b \\ 0 \end{pmatrix}$ (de norme c^2+b^2)

$$\begin{aligned}
\cos\theta &= \frac{1}{c^2+b^2}\Big[(-c)\cdot\Big((1-2c^2-2d^2)(-c)+(2bc-2ad)(b)\Big) \\
&\quad + (b)\cdot\Big((2bc+2ad)(-c)+(1-2d^2-2b^2)(b)\Big)\Big] \\
&= \frac{1}{c^2+b^2}\Big[c^2(1-2c^2-2d^2)-bc(2bc-2ad) \\
&\quad - bc(2bc+2ad)+b^2(1-2d^2-2b^2)\Big] \\
&= \frac{1}{c^2+b^2}\Big[c^2-2c^2c^2-2c^2d^2-2b^2c^2+2abcd \\
&\quad - 2b^2c^2-2abcd+b^2-2b^2d^2-2b^2b^2\Big] \\
&= \frac{1}{c^2+b^2}\Big[c^2+b^2-2c^2(c^2+d^2+b^2)-2b^2(b^2+c^2+d^2)\Big] \\
&= \frac{1}{c^2+b^2}\Big[(c^2+b^2-2c^2(1-a^2)-2b^2(1-a^2)\Big] \\
&= (2a^2-a^2-b^2-c^2-d^2) \\
&= (2a^2-1)
\end{aligned}$$

ce qui peut aussi s'écrire

$$\cos\frac{\theta}{2} = a.$$

...récapitulons

A un quaternion $r = a+bi+cj+dk$ de norme 1 on peut associer la transformation R_r qui à tout quaternion $\xi = t+xi+yj+zk$ fait correspondre le quaternion $R_r(\xi) = r\xi\overline{r}$.

On a les propriétés:

- $R_r(\lambda\xi + \lambda'\xi') = \lambda R_r(\xi) + \lambda' R_r(\xi')$
- $R_{rr'}(\xi) = R_r(R_{r'}(\xi))$
- $R_r(t + 0i + 0j + 0k) = t + 0i + 0j + 0k$
- Les composantes imaginaires pures (x', y', z') de $\xi' = R_r(\xi)$ sont obtenues à partir des composantes imaginaires pures (x, y, z) de ξ par le produit matriciel

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = M \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

où M est la matrice 3×3

$$M = \begin{pmatrix} 1 - 2c^2 - 2d^2 & 2bc - 2ad & 2bd + 2ac \\ 2bc + 2ad & 1 - 2d^2 - 2b^2 & 2cd - 2ab \\ 2bd - 2ac & 2cd + 2ab & 1 - 2b^2 - 2c^2 \end{pmatrix}$$

M est une matrice de rotation d'angle θ (avec $cos\frac{\theta}{2} = a$) autour d'un axe passant par l'origine de coefficients directeurs (b, c, d).

- Si M et M' sont les deux matrices de rotations associées aux quaternions r et r' alors MM' est la matrice de rotation associée au quaternion rr'.

5.3.3 De la rotation au quaternion

Réciproquement, à toute rotation de ^{3}, d'axe passant par l'origine, peut être associé un quaternion de norme 1.

Soit $\Delta = \begin{pmatrix} \delta_x \\ \delta_y \\ \delta_z \end{pmatrix}$ et θ l'axe et l'angle d'une rotation dans ^{3} (avec $\delta_x^2 + \delta_y^2 + \delta_z^2 = 1$).

On lui associe le quaternion

$$r = \cos\frac{\theta}{2} + \sin\frac{\theta}{2}\delta_x i + \sin\frac{\theta}{2}\delta_y j + \sin\frac{\theta}{2}\delta_z k$$

Ce quaternion existe toujours et est de norme 1.

En effet

$$\begin{aligned} \|r\| &= \cos^2\frac{\theta}{2} + \sin^2\frac{\theta}{2}\delta_x^2 + \sin^2\frac{\theta}{2}\delta_y^2 + \sin^2\frac{\theta}{2}\,\delta_z^2 \\ &= \cos^2\frac{\theta}{2} + \sin^2\frac{\theta}{2}(\delta_x^2 + \delta_y^2 + \delta_z^2) \\ &= \cos^2\frac{\theta}{2} + \sin^2\frac{\theta}{2} \\ &= 1 \end{aligned}$$

Conclusion

Il existe un isomorphisme entre les quaternions de norme 1 et les rotations d'axe passant par l'origine de ^{3}.

Au produit des rotations est associé le produit des quaternions.

Toutes les opérations sur les rotations peuvent être faites en utilisant les opérations de l'algèbre des quaternions.

5.3.4 Ordre d'un produit de matrices et rotations

En 5.1.1 nous avions affirmé que l'ordre d'un produit de matrices de rotation définit si ces rotations sont faites autour d'axes fixes ou d'axes mobiles (entrainés dans les rotations); la définition d'une rotation en termes de quaternions nous permet de démontrer cela fort élégamment de la manière suivante:

Soit Oz l'axe d'une rotation d'angle α_x et Oy l'axe d'une rotation d'angle α_y.

Soit Oy' l'axe obtenu par rotation de Oy autour de Oz de l'angle α_z.

On a, en notant Z, Y et Y' les quaternions associés aux rotations d'axes respectifs Oz, Oy et Oy' et en assimilant ces axes aux mêmes quaternions

$$Y' = Z\,Y\,\overline{Z}.$$

La rotation d'un point quelconque ξ s'écrit

$$\begin{aligned} R_{y'}\,R_z(\xi) &= Y'Z\,\xi\,\overline{Z}\,\overline{Y'} \\ &= Z\,Y\,\overline{Z}\,Z\,\xi\,\overline{Z}\,\overline{Z\,Y\,\overline{Z}} \end{aligned}$$

$$\begin{aligned} &= Z\,Y\,\overline{Z}\,Z\,\xi\,\overline{Z}\,\overline{\overline{Z}}\,\overline{Y}\,\overline{Z} \\ &= Z\,Y\,\overline{Z}\,Z\,\xi\,\overline{Z}\,Z\,\overline{Y}\,\overline{Z} \\ &= ZY\xi\overline{YZ} \\ &= R_z R_y(\xi). \end{aligned}$$

5.3.5 Puissance d'un quaternion

Nous venons de voir que tout quaternion de norme 1 peut s'écrire

$$q = \cos\alpha + \sin\alpha A$$

où $A = a_x i + a_y j + a_z k$ tel que $\|A\| = 1$.

On vérifie facilement que

$$\begin{aligned} q^2 &= \cos 2\alpha + \sin 2\alpha A \\ q^3 &= \cos 3\alpha + \sin 3\alpha A \\ &\dots \\ q^n &= \cos n\alpha + \sin n\alpha A \end{aligned}$$

de même

$$\begin{aligned} q^{-1} &= \cos(-\alpha) + \sin(-\alpha)A \\ &= \cos\alpha - \sin\alpha A \end{aligned}$$

d'une manière générale on définit pour tout $u \in$

$$q^u = \cos u\alpha + \sin u\alpha A.$$

Si l'on considère q comme étant une rotation d'axe A d'angle $\theta = 2\alpha$, q^u est la rotation de même axe A d'angle $u\theta$.

5.4 Interpolation

Nous pouvons aborder maintenant le problème de l'interpolation de ces rotations en utilisant les propriétés des quaternions [Shoemake 1985, Le Borgne 1987, Yahia et Gagalowicz, 1989].

5.4.1 De q_1 à q_2

Etant données deux orientations définies par les quaternions de norme 1, q_1 et q_2, comment passer de l'une à l'autre de façon continue [Yahia et Gagalowicz, 1989, Shoemake 1985]?

Il faut définir $q(u)$, de norme 1 pour tout u, tel que

$$q(0) = q_1 \qquad\qquad q(1) = q_2$$

et tel que $q(u)$ décrive un chemin "minimal" de q_1 à q_2.

Géométrie sur la sphère de ^{4}

Ce chemin "minimal" suit la géodésique passant par ces deux points [Le Borgne 1987]. Puisque les quaternions de norme 1 sont situés sur $S3$, la sphère unité de ^{4}, $q(u)$ parcourt l'**arc de grand cercle** passant par q_1 et q_2 et ce de manière que les angles entre q_1 et $q(u)$ d'une part et $q(u)$ et q_2 d'autre part soient tels que

$$u \times \widehat{q_1 q(u)} = (1-u) \times \widehat{q(u) q_2}.$$

En notant

$$\alpha_{12} = \widehat{q_1 q_2} \qquad \alpha_{1u} = \widehat{q_1 q(u)} \qquad \alpha_{u2} = \widehat{q(u) q_2}$$

il suffit de prendre

$$\alpha_{1u} = (1-u)\alpha_{12} \qquad\qquad \alpha_{u2} = u\alpha_{12}$$

on a, en notant "$\cdot$" le produit scalaire,

$$\cos\alpha_{12} = q_1 \cdot q_2 \qquad \cos\alpha_{1u} = q_1 \cdot q(u) \qquad \cos\alpha_{u2} = q(u) \cdot q_2.$$

Il ne reste plus qu'à trouver a et b, fonctions de u tels que

$$\begin{cases} q(u) = aq_1 + bq_2 \\ q_1 \cdot (aq_1 + bq_2) = (1-u) q_1 \cdot q_2 \\ (aq_1 + bq_2) \cdot q_2 = u q_1 \cdot q_2 \\ \|aq_1 + bq_2\| = 1 \end{cases}$$

La première équation indique que $q(u)$ est combinaison linéaire de q_1 et q_2. La deuxième et troisième expriment la relation concernant les angles, la quatrième, que le quaternion doit être de norme 1.

Les quaternions q, q_1, q_2 sont dans un plan passant par l'origine, l'intersection de ce plan avec la sphère unité (condition 4) est le grand cercle passant par les trois points. Si $\alpha_{12} \neq 0$ et $\alpha_{12} \neq \pi$ (q_1 et q_2 non confondus ni opposés), la solution est unique et s'écrit

$$q(u) = \frac{\sin((1-u)\alpha_{12})}{\sin\alpha_{12}} q_1 + \frac{\sin((u)\alpha_{12})}{\sin\alpha_{12}} q_2.$$

En effet, montrons par exemple que $\widehat{q_1 q(u)} = q_1 \cdot q(u) = \cos((u)\alpha_{12})$:

$$\begin{aligned}
q_1 \cdot q(u) &= \frac{\sin((1-u)\alpha_{12})}{\sin\alpha_{12}} q_1 + \frac{\sin((u)\alpha_{12})}{\sin\alpha_{12}} q_2 \\
&= \frac{\sin((1-u)\alpha_{12})}{\sin\alpha_{12}} q_1 \cdot q_1 + \frac{\sin((u)\alpha_{12})}{\sin\alpha_{12}} q_1 \cdot q_2 \\
&= \frac{1}{\sin\alpha_{12}} [\sin((1-u)\alpha_{12}) + \frac{1}{2}(\sin((1+u)\alpha_{12}) - \sin((1-u)\alpha_{12}))] \\
&= \frac{1}{2\sin\alpha_{12}} [\sin((1-u)\alpha_{12}) + \sin((1+u)\alpha_{12})] \\
&= \frac{1}{\sin\alpha_{12}} [\sin\alpha_{12}\cos((u)\alpha_{12})] \\
&= \cos((u)\alpha_{12}),
\end{aligned}$$

on vérifie de même que $q(u) \cdot q_2 = \cos((1-u)\alpha_{12})$ et que $q(u)$ est de norme 1.

Le groupe multiplicatif des quaternions de norme 1

Soient p_1 et p_2 deux points de ^{n}, l'interpolation linéaire, chemin le plus court de p_1 à p_2 est donnée par $p(u) = (1-u)p_1 + up_2$. Par similitude, en remplaçant l'addition + par le produit de quaternions, la multiplication par un scalaire par l'élévation à la puissance, l'interpolation *linéaire* du quaternion q_1 à q_2 s'écrira $q(u) = q_1^{1-u} q_2^u$, ou, plus exactement en tenant compte du fait que le produit de quaternions n'est pas commutatif:

$$q(u) = q_1 (q_1^{-1} q_2)^u.$$

On peut montrer que

$$q_1 (q_1^{-1} q_2)^u = \frac{\sin((1-u)\alpha_{12})}{\sin(\alpha_{12}} q_1 + \frac{\sin((u)\alpha_{12}}{\sin\alpha_{12}} q_2.$$

5.4.2 Interpolation d'une suite de quaternions

Avec ce que nous venons de définir est-il possible d'adapter les méthodes d'interpolations décrites au chapitre 4?

Splines cubiques naturelles ...*la bonne manière*

L'équation du paragraphe 4.2.1

$$x(u) = a_{3_x}u^3 + a_{2_x}u^2 + a_{1_x}u + a_{0_x}$$

peut s'écrire

$$x(u) = u(u(u(a_{3_x}) + a_{2_x}) + a_{1_x}) + a_{0_x}$$

et transposée aux quaternions,

$$x(u) = (((a_{3_x})^u a_{2_x})^u a_{1_x})^u a_{0_x}.$$

Il faudrait calculer la dérivée par rapport à u, sachant que les produits de quaternions ne sont pas commutatifs. Il faudrait d'autre part définir un ensemble de notations où la multiplication par une matrice n'est pas à considérer comme une combinaison linéaire mais comme un produit de termes élevés à une puissance ...que signifie alors l'inversion de matrice?

Ne sachant pas résoudre ces problèmes, nous adopterons une solution plus simple, basée sur des interpolations sur la sphère de ^{4}.

Interpolation sur la sphère de ^{4}

Nous avons vu en 5.4.1 comment aller de q_1 à q_2 en suivant le grand cercle passant par ces deux quaternions. L'interpolation, par splines naturelles cubiques d'une suite de quaternions *rendez-vous*, est calculée d'après la méthode mise au point dans la partie 4.3 du chapitre 4.

Dans un premier temps il faut calculer la direction des tangentes (en fait les grands cercles supportant ces tangentes) puis choisir les points r_{i-1} et l_i pour appliquer la méthode telle qu'elle a été décrite en 4.3.5.

Dans ^{n} la dérivée D_i se calcule (cf. 4.3.4) par

$$D_i = 3 \sum_{k=1}^{+\infty} -(\sqrt{3} - 2)^k (p_{i+k} - p_{i-k}).$$

Peut-on appliquer directement ces formules aux quaternions?

Elle devraient correspondre à des additions d'arcs de grands cercles sur la sphère de ^{4},... mais, de même que les interpolations linéaires

$$(1 - u)p_1 + up_2$$

sont remplacées par

$$\frac{\sin((1-u)\alpha_{12})}{\sin \alpha_{12}} q_1 + \frac{\sin((u)\alpha_{12})}{\sin \alpha_{12}} q_2,$$

il faudrait remplacer les termes $p_{i+k} - p_{i-k}$ par la combinaison correspondante des quaternions.

Ne disposant pas d'expressions mathématiques des contraintes qui doivent être repectées nous ne pouvons pas définir clairement quelle solution choisir. La première question est comment déterminer la meilleure tangente en un point p_i à la trajectoire passant par trois points p_{i-1}, p_i et p_{i+1} de $S3$? Est-ce la tangente au petit cercle passant par les trois points, la bissectrice extérieure, en p_i, du triangle sphérique (à quatre dimensions) formé par les trois points, etc.? Une autre question est d'associer à ces sommes une signification correcte pour le calcul des quaternions. La question est ouverte et ne pourra être tranchée que par la définition de contraintes précises.

Nous n'avons pas encore eu le temps de nous attaquer à ce problème et avons choisi de calculer les tangentes comme bissectrices extérieures des triangles. Les résultats que nous obtenons sont satisfaisants.

La figure 5.1 donne une représentation du résultat sous la forme d'un objet graphique composé par l'ensemble des axes de rotations dans ^{3}, leur longueur (et la couleur si l'image est en couleur) de chaque axe symbolisant l'angle de rotation.

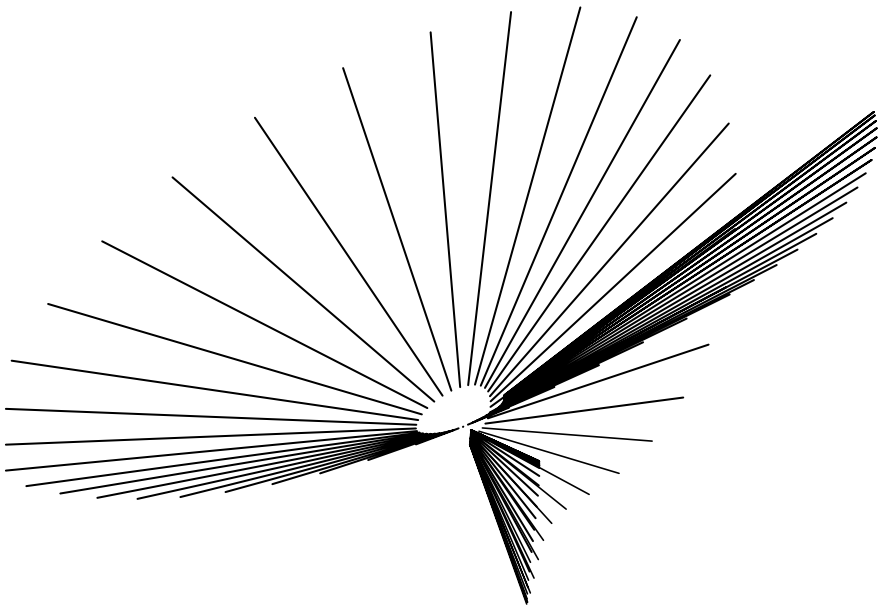


Figure 5.1: Axes de rotations dans ^{3}, après interpolation. A chaque axe est associé une longueur symbolisant l'angle de rotation.

Nous avons mis au point, dans *A*GRAPH, une méthode permettant d'animer cet ensemble d'axes par une suite de rotations qui amène à chaque instant l'axe, représentant l'orientation courante, perpendiculaire à l'écran. L'effet est trés esthétique et permet de juger sur écran de la qualité de l'interpolation.

5.5 Autres applications des quaternions

La modélisation des rotations par des quaternions a d'autres applications parmi lesquelles nous citerons la modélisation de surfaces [Pletincks 1988], la commande de rotations d'objets dans l'espace [Le Borgne 1987], la recherche de l'orientation de molécules [Mackay 1983] et plus généralement, les nombreux domaines abordés par de Casteljau [de Casteljau 1987].

Chapitre 6

Applications

6.1 Introduction

Nous donnons ici un bref aperçu des applications que nous avons développées sur ou pour la station *PS300* Evans & Sutherland.

Certaines sont des applications directes de *A*GRAPH, d'autres en sont plus ou moins dérivées. L'un des points communs est l'utilisation quasi générale d'objets graphiques dans lesquels intervient une dimension supplémentaire qui peut être le temps pour des objets animés ou l'instanciation multiple.

Le *PS300* étant avant tout un outil de visualisation qui permet l'interactivité, il s'est vite révélé comme outil indispensable: les *besoins* apparaissant au fur et à mesure.

Certaines des applications présentées ci-dessous, ont été réalisées afin d'illustrer certains aspects de la programmation du *PS300*.

6.2 AGRAPH

*A*GRAPH permet la réalisation de séquences d'animation destinées à être filmées, ou simplement regardées. En dehors d'un support pellicule ou vidéo, ces séquences peuvent servir d'échange d'informations entre laboratoires: il suffit d'envoyer, par réseau, les fichiers de scènes clés au destinataire disposant du logiciel *A*GRAPH qui pourra recalculer les images intermédiaires puis admirer le film sur son écran.

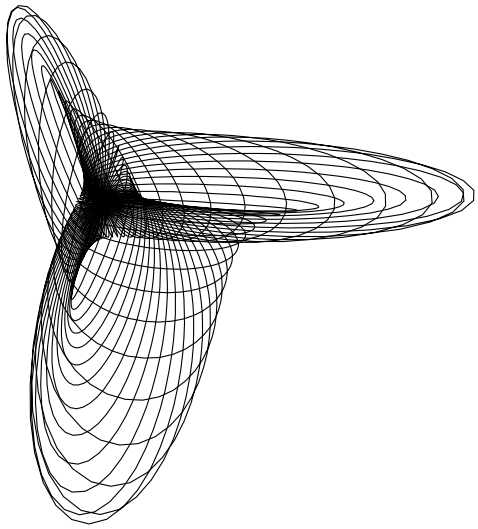


Figure 6.1: La surface de BOY.

6.2.1 Surface de Boy

Avec François Apéry, nous avons entrepris l'étude et la visualisation d'un grand nombre de surfaces de ^{3} données par des équations polynômiales ou paramétrées par des fractions rationnelles. Ce travail [Ripp 1987] a permis à F. Apéry d'illustrer magnifiquement son ouvrage *Models of the Real projective Plane* [Apéry 1987].

Nous avons également entrepris la réalisation d'une séquence d'animation avec *A*GRAPH ayant trait à une de ces surfaces, la surface de Boy. Celle-ci est une représentation du plan projectif comme surface immergée dans ^{3} ne comportant pas de singularité [Apéry 1986]. Cette surface fermée, non orientée, est une des étapes du retournement de la sphère [Morin et Petit, 1979] (une autre surface [Morin 1978] topologiquement identique à celle de Boy est représentée figure 6.2). Elle possède un point triple et une courbe d'autointersection.

Généralement nous avons représenté les surfaces comme un ensemble de courbes, chacunes d'elle étant formées d'une suite de segments de droite. La surface de Boy de la figure 6.1 est composée de 60 ellipses composées de 60 segments.

Le film doit permettre de comprendre la topologie de la surface. La première partie de l'animation montre une construction de la surface à l'aide des ellipses, des mouvements de caméra, agrandissements, *clippings*, mise en valeur de la courbe

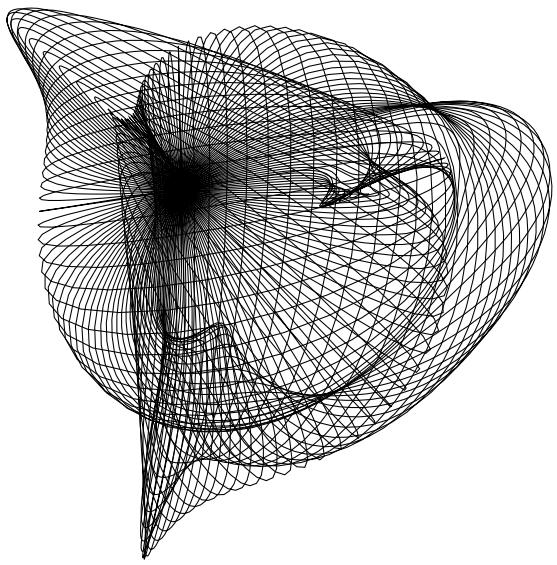


Figure 6.2: Retournement de la sphère par B.MORIN.

d'autointersection et du point triple, permettent de *visiter* la surface et de l'*expliquer*. La deuxième partie montre la déformation d'une surface de Steiner (aussi appelée *Romaine*), qui est également une immersion du plan projectif dans ^{3}, mais avec singularités, en une surface de Boy (sans singularité). Pour cette séquence, les objets graphiques, qui sont des listes de vecteurs, associés à chaque étape de la déformation ont du être calculés séparément, ne pouvant pas être obtenus par les transformations géométriques simples de *A*GRAPH. La visualisation d'une liste de vecteurs données, parmi toutes celles présentes en mémoire est faite par l'intermédiaire d'une structure `level_of_detail` présentée en A.6.7 (dans [Amerein et Spehner, 1989] est présentée une application de cette technique). La séquence d'animation peut néanmoins être faite par une mise en scène classique, nous avons créer pour cela un type d'acteur *spécialisé* pour lequel le bouton *contraste* est associé au numéro de l'image à afficher.

Nous avons effectué un enregistrement à l'aide d'une caméra mécanique 16mm Bolex. Le déclenchement manuel *vue par vue* a pu être automatisé et commandé par la station graphique grâce à une interface aimablement conçue et développée par Serge Wendling. Le tournage par cette caméra a nécessité des expositions vue par vue pour avoir une luminosité suffisante. Nous avons dû programmer un réseau de fonctions cadencé par les données permettant de commander le temps d'exposition, les arrêts nécessaires au remontage de la caméra ou au changement de pellicule, ou au

chargement de la station graphique par les images des surfaces qui étant composées de beaucoup de vecteurs (entre 5 et 10 000) causaient des remplissages de mémoire. Les arrêts en cours d'exposition pouvant survenir n'importe quand, sur commande manuelle (changement de pellicule) ou automatique (surface nécessaire non chargée) ce réseau doit pouvoir arrêter après exposition de l'image courante.

Le film 16mm a été sonorisé par F. Apéry et tiré sur cassette vidéo.

6.2.2 Hologramme de la molécule de tRNA

Les techniques holographiques permettent maintenant la création d'hologrammes à partir d'images synthétisées par ordinateurs [Neal 1988]. C'est ainsi que nous avons pu réaliser un hologramme cylindrique représentant une molécule d'acide ribonucléique de transfert [Westhof *et al.*, 1985]. La société HOLOGRAMME INDUSTRIE a créé l'hologramme à partir d'un film de 1080 images que nous avons enregistrées avec une caméra Beaulieu 16mm (aimablement prêtée par M. Roparts de l'Institut de Psychologie de l'ULP). Cet hologramme ainsi que celui de la couverture du mensuel *Biofutur* a pu être mené à bien grâce aux efforts de Marcel Boeglin du laboratoire de cristallographie de l'IBMC et de Michel Mellet de la société NEOSYSTEM.

Les 1080 images représentent la molécule tournant sur son axe vertical à raison d'un tiers de degré par image. Le résultat, bien que satisfaisant, mériterait, là aussi, une meilleure qualité de prise de vue.

6.2.3 Hologramme de la couverture du mensuel *Biofutur*

Cet hologramme imprimé, tiré à 12 000 exemplaires [Hologramme 1987] est un hologramme de la molécule de cardiotoxine de venin de serpent [Rees *et al.*, 1990]. La couverture de *IEEE Computer Graphics & Applications* [Neal 1988] a été réalisée suivant les mêmes procédés.

L'enregistrement des 200 images de cet hologramme a été fait par une caméra professionnelle 35mm pour obtenir cette fois une meilleure qualité d'image, l'hologramme devant être imprimé.

Il a nécessité une prise de vue particulière qui consiste à utiliser la projection `eye_back` et à déplacer le point d'où l'on regarde, `look from`, ainsi que les paramètres

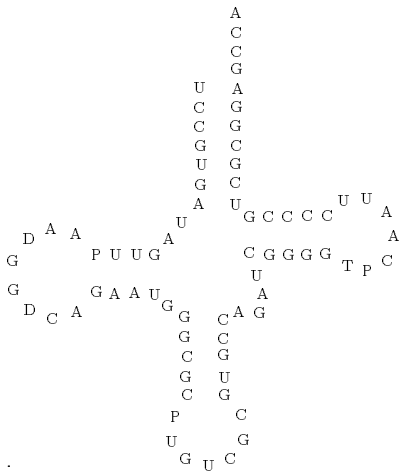


Figure 6.3: Séquence et structure secondaire en feuille de trêfle du $tRNA^{Asp}$ de levure.

de `eye_back` calculés en conséquence, ceci afin d'obtenir des vues spécifiques pour ce type d'hologramme plan.

6.2.4 Film didactique sur les molécule de tRNA

Nous avons réalisé un film pédagogique destiné à présenter les molécules d'acides ribonucléiques de transfert [Moras *et al.*, 1980]. Pour cela nous avons développé une série de programmes permettant de visualiser et de manipuler la structure tertiaire (coordonnées tridimensionnelles, fig. 6.4), secondaire (feuille de trêfle, fig.6.3) et primaire (la séquence) de ces molécules. En liaison avec les programmes de calcul de ces structures, tels que *Frodo* [Jones 1982], ils permettent de représenter en même temps sur l'écran, les trois types d'informations. A l'aide d'un bouton, l'utilisateur peut mettre en valeur différentes zones de la molécule, à la fois sur les structures 3D, 2D et 1D. De plus à chaque zone peut être associé un texte (écrit par l'utilisateur). Parallèlement à ce choix de zones à afficher, on peut manipuler la structure tridimensionnelle et la faire évoluer dans l'espace afin de la *montrer*. Toutes ces opérations sont prises en compte par *A*GRAPH et peuvent fournir la trame d'une animation.

Nous avons ainsi créé toutes les séquences d'un film complet avec générique, visualisation 3D et affichage de texte explicatif évoluant sur l'écran (le texte lui-même ainsi que les structure 2D et 1D étant des acteurs *A*GRAPH). La qualité de

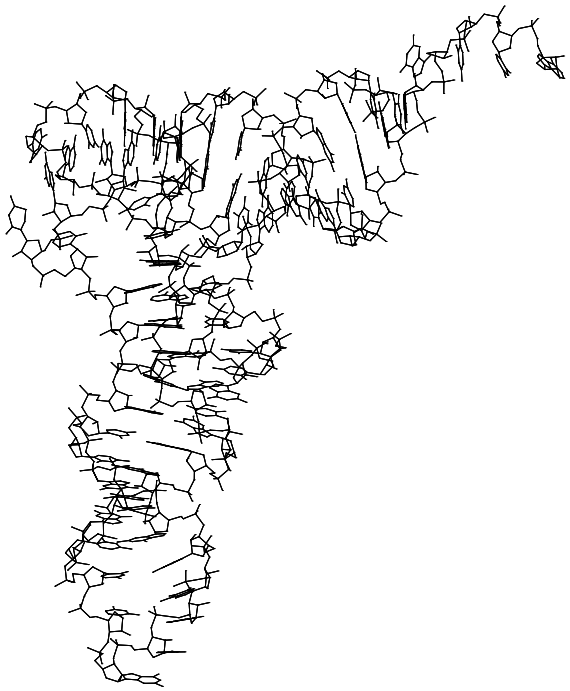


Figure 6.4: Structure tertiaire du $tRNA^{Asp}$ de levure.

l'enregistrement sur film 16mm par la caméra Bolex n'a malheureusement pas permis sa diffusion. Béatrice Amerein a parfait ce travail en l'intégrant au logiciel *Frodo* . Il est maintenant accessible en dehors de *A*GRAPH à la communauté cristallographique et est largement utilisé [Amerein 1988].

6.2.5 Phipsi

Ce programme construit autour de *A*GRAPH permet la manipulation et la visualisation de molécules de protéine [Ripp *et al.*, 1991]. Son intérêt réside dans la possibilité de manipuler interactivement en temps réel les angles dièdres ϕ et ψ définissant la structure tridimensionnelle de ces molécules. Son aspect pédagogique a été particulièrement développé, il est en effet possible de voir les conformations adoptées par la molécule pendant que l'on déplace le crayon optique sur le diagramme de Rahmachandran, représentation graphique bidimensionnelle de l'ensemble des conformations possibles (fig. 6.5).

La suite des conformations choisies par l'utilisateur pouvant être mémorisées

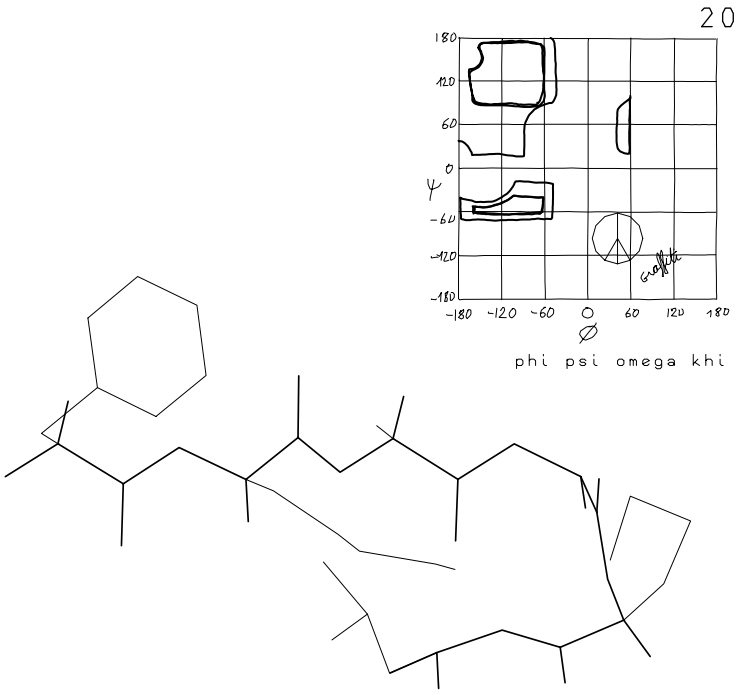


Figure 6.5: Image affichée par Phipsi. En désignant un point du diagramme de Rahmachandran représenté en haut à droite de l'écran, on fait adopter au polypeptide la conformation correspondant aux angles cliqués.

comme lors d'une mise en scène, il est possible de réaliser des séquences d'animation.

6.2.6 Halley

*A*GRAPH nous permis de modéliser les mouvements des planètes et astéroïdes du système solaire en particulier de la comète de Halley. Les plans des orbites et les périodes de révolutions sont respectés. L'utilisateur peut visualiser à sa guise n'importe quelle région du système solaire. Il peut voir défiler les planètes, suivre l'une quelconques d'entre elles (en y *reliant* la position de l'observateur `look from`).

6.3 MoPs

MoPs est un logiciel de visualisation et de manipulation de molécules. Il a été écrit de telle sorte qu'il puisse tourner sur le *PS300* sans l'utilisation de l'ordinateur hôte (mis à part le chargement du programme). Toutes les commandes sont entièrement exécutées par le réseau Data Driven sans échange avec des programmes de l'hôte. Il est en particulier possible de charger l'application par l'intermédiaire d'une disquette (pour des démonstrations par exemple).

En plus des opérations de visualisation, il permet à l'utilisateur qui désigne un atome, représenté comme un ellipsoïde, de connaitre ses coordonnées, sa nature et son nom (ceci est fait par la recherche dans une liste comme nous l'avons indiquer en 3.4.2). *MoPs* prend en compte les opérations de symétrie cristallographiques de la molécule, affichées ou non au choix de l'utilisateur. Il permet de couper des liaisons ou d'en créer de nouvelles. Il calcule les distances interatomiques, les angles de valence et les angles dièdres. C'est un programme indépendant non relié à *A*GRAPH.

6.4 ROC et ROCI

Le problème que doit résoudre un critallographe disposant d'un cliché de diffraction aux rayons X d'une molécule biologique, est de déterminer à quelle orientation du cristal ce cliché correspond [Blundel et Johnson, 1976]. Une première méthode consiste à calculer à partir d'une orientation donnée (présumée), l'image que ce cristal, dont on connait plus ou moins précisément les paramètres de maille, doit créer

lors de son exposition aux rayons X et à comparer cette image *calculée* avec l'image *réelle* visible sur le cliché. Cette comparaison des deux images peut se faire à l'écran du *PS300* grâce au logiciel *ROC*. Après un certain nombre d'essais-erreurs, et beaucoup de métier, le cristallographe arrive généralement à affiner cette orientation. *ROC* facilite cette tâche en offrant des outils interactifs de visualisation pratiques et efficaces: mémorisation d'images, choix aisé de nouveaux angles d'orientation, détermination automatique de certains paramètres, etc..

Le calcul d'une image correspondant à une orientation donnée est fait sur l'ordinateur hôte et nécessite plusieurs secondes, ce qui interdit le temps réel. C'est pour cela que nous avons mis au point le logiciel *ROCI*, basé sur une idée originale de Philippe Dumas, qui autorise le temps réel [Dumas et Ripp, 1986]. On peut montrer simplement que l'image d'un cliché de diffraction correspond géométriquement à la projection de centre C sur un plan P (fig. 6.6) de l'ensemble des points d'un réseau tridimensionnel périodique (déterminé uniquement par les valeurs des paramètres de maille du cristal) situés *près* de la sphère de centre C et de rayon R ($= 1/\lambda$, λ longueur d'onde) Une légère oscillation du réseau autour du point O, intersection de la sphère et de l'axe des rayons X, fait passer les points du réseau proches de la sphère à travers celle-ci, ce sont les projections de ces points qui sont visibles sur le cliché de diffraction.

Effectuons une inversion de pôle O de puissance ρ. On peut déterminer $\rho = \overline{OP} \times \overline{OH}$ de sorte que l'image de la sphère par cette inversion soit le plan P. Si l'on soumet tous les points du réseau à cette inversion, les points qui étaient proches de la sphère se retrouveront proche du plan P. Un clipping parallèle à P en deça et au delà de P donnera une certaine image du réseau inversé. Cette image doit correspondre à l'image du cliché expérimental auquel on aura fait subir une transformation proche d'une inversion (voir figure 6.6). On remarquera qu'une rotation du cristal autour du point C correspond très exactement à la même rotation du réseau et du réseau inversé (l'inversion d'une rotation est la rotation de l'inversion, s'ils sont tous deux de même centre, ce qui est le cas).

ROCI est un bel exemple de graphisme interactif qui profite de la puissance de la machine (ici le clipping).

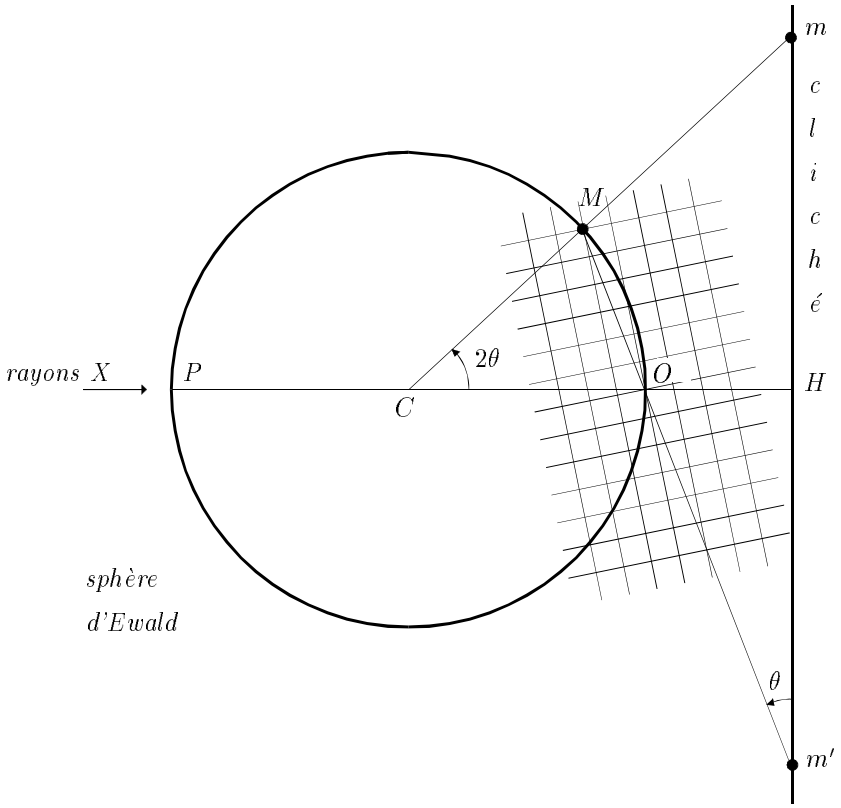


Figure 6.6: *ROCI*: Règlage de l'Orientation d'un Cristal par Inversion. Un point M du réseau périodique proche de la sphère d'Ewald de centre C forme sur le cliché une tache de diffraction m. L'inversion de pôle O de puissance $\rho = \overline{OP} \times \overline{OH}$ transforme la sphère en un plan qui est le plan du cliché. Le point m', inverse de M, est relié à m par $\overline{Hm'} = \rho/[\overline{Hm}(1-\tan^2\theta)]$. On peut par cette transformation, calculer l'image de l'inverse du cliché.

6.5 Interpolation dans TEX

Ce document a été réalisé à l'aide du traitement de texte TEX [Knuth 1986] et LATEX [Lamport 1986]. Ce logiciel s'il est parfait pour son aspect typographique a de sérieuses lacunes pour tout ce qui concerne les dessins. Il est en effet trés difficile de faire des figures composées de segments de droite, et pratiquement impossible de tracer des courbes. Afin d'obtenir un document TEX *pur* comportant tous ces dessins nous avons développé une interface permettant de créer des ordres TEX à partir de commandes habituelles de dessin, nous y avons aussi implanté les algorithmes d'interpolation pour obtenir des courbes lisses à partir de points *rendez-vous*.

D'autre part nous avons mis au point une série d'interfaces permettant d'inclure dans un document TEX presque tous les types de documents graphiques, à des fins d'analyse ou de publications [Mourey *et al.*, 1990]. Tous les programmes développés dans *A*GRAPH comportant une sortie graphique peuvent ainsi fournir aisément une sortie directement compréhensible par TEX.

Chapitre 7

Conclusion

7.1 L'animation graphique interactive est un outil

Dans ce manuscrit nous résumons l'ensemble des travaux ayant trait à la station graphique *PS300* Evans & Sutherland. Certaines études ont été abordées dans le but de résoudre un problème particulier, elles ont abouti à des réalisations et nécessité certains développements qui eux-mêmes ont débouché sur d'autres études. Il nous était difficile de prévoir quelles applications devaient être développées, ne possédant aucune pratique ni expérience en la matière. En effet, **les besoins apparaissent avec les outils**, ensuite d'autres outils sont créés en fonction de ces nouveaux besoins. Le graphisme, l'animation graphique et l'interactivité sont vus, d'abord, comme des gadgets, puis deviennent habitudes, pour finalement être nécessaires. La plupart des applications possibles restent à découvrir, l'animation graphique interactive en est encore au stade où elle consiste à simuler certaines de nos pratiques courantes, elle ne tire pas pleinement profit des nouvelles possibilités qu'offre la machine pour visualiser et manipuler des entités abstraites.

*A*GRAPH et la station graphique nous sont apparus en fait comme un outil qui permet de fouiller et de **visualiser des données**: pour les banques de données de structures de molécules, par exemple, il est toujours nécessaire d'utiliser une représentation graphique pour y *voir* quelque chose, nous avons développé *Ecosse*, *Prot* pour les comparaisons de structures primaires, *Prot*, *Frag*, pour l'étude des

structures secondaires, ou *Puck*, *A*GRAPH, pour l'analyse fine des structures tridimensionnelles [Amerein *et al.*, 1987, Amerein 1988]. D'autres programmes (*Packing* [Moras et Bergdoll, 1988]) permettent de visualiser l'empilement cristallin de ces structures.

*A*GRAPH est aussi un **outil pédagogique**, il permet, par exemple avec *PhiPsi*, de montrer ce qu'est une hélice de protéine, d'expliquer la recombinaison des ADN [Moulinier 1990].

La station graphique *PS300* est une loupe que l'on déplace sur les informations existantes. Mais, sans outil particulier, *seul* celui *qui sait* programmer, *sait* visualiser les données pour les comprendre. Nous aimerions, par contre, que cela soit **possible pour tout le monde**, nous voudrions offrir un outil simple qui permette de le faire: les applications qui ont été développées vont dans ce sens.

L'ordinateur est devenu l'**outil nécessaire** à la plupart des chercheurs. Nous pensons qu'il en sera bientôt de même pour l'**animation graphique interactive**. En déplaçant cette *loupe* sur les informations stockées sur les disques. On verra défiler, ici des textes, là des graphiques, puis des structures tridimensionnelles d'objets réels mais on pourra aussi comprendre comment elles évoluent dans le temps ou comment elles réagissent aux actions qu'on leur impose. Souvent dans un livre figurent les conseils pour s'en servir, comment profiter pleinement de son contenu. Pourquoi les informations stockées dans les ordinateurs ne seraient-elles pas aussi conviviales? Quand s'arrêtera-t-on de considérer une information comme le contenu d'un fichier? Quand décidera-t-on que cette information ne peut être comprise que si elle est analysée correctement par des programmes de visualisation adéquats?

7.2 Un outil à améliorer

Nous avons développé un outil qui nous fournit quelques réponses à ces questions.

Il n'est évidemment qu'un prototype et demande à être réécrit et amélioré. Les solutions que nous avons adoptées permettent d'offrir à l'utilisateur un environnement relativement simple et convivial.

En plus des outils d'interaction, il faudra offrir à l'utilisateur des moyens de construire simplement des objets graphiques complexes et, si les performances des machines le permettent un jour, la possibilité de manipuler des **objets graphiques déformables** qui ne soient plus seulement filaires mais surfaciques, ceci directement, ou par l'intermédiaire de programmes tournant en temps réel sur de puissants ordinateurs.

L'utilisation des techniques d'**interpolation** qui rendent les mouvements aussi lisses que possibles ne sont peut être pas les mieux adaptées mais semblent être les plus simples et les moins contraignantes. La méthode d'interpolation par approximation des tangentes que nous avons mise au point permet de calculer les interpolations sans avoir à prendre en compte pour chaque portion de la courbe l'ensemble des points de rendez-vous. Elle évite les calculs sur de grandes matrices dont on ne connait pas à priori les dimensions. Nous avons pu étendre cette méthode à l'interpolation en fonction du temps moyennant des manipulations de matrices de petites dimensions fixes. Actuellement les temps de calcul sont encore largement négligeables par rapport au temps nécessaire à l'envoi des valeurs interpolées de l'ordinateur vers la station graphique. Des développements sont encore nécessaires pour la mise au point d'algorithmes performants permettant le temps réel. Ceci suppose évidemment l'implantation de ces algorithmes directement sur la station graphique. Leur réalisation par des réseaux cadencés par les données, qui est possible, serait particulièrement intéressante.

Une étude approfondie des **quaternions** mérite d'être entreprise. Les références bibliographiques traitant de leurs utilisations dans des cas simples sont peu nombreuses, souvent considérées comme des cas particuliers de théories plus générales et sont de ce fait rarement utilisées. Le contrôle des rotations dans l'espace, qui peut sembler simple à priori, conduit en fait à des calculs très compliqués et soulève des problèmes géométriques qui nous sont peu familiers.

L'une des difficultés que nous avons rencontrées a été de définir des suites de rotations qui soient le moins "oscillantes" possibles: une rotation définie par un axe et un angle admet plusieurs représentations, le sens de l'axe pouvant être positif

ou négatif, suivant la valeur de l'angle qui lui-même est défini à 2π près. Si les suites des orientations-clés ne sont pas choisies correctement, les interpolations, quelques soient la méthode utilisée, donnent des résultats désastreux. L'utilisation des quaternions n'apporte pas de réponse, l'interprétation d'un quaternion comme rotation n'est pas univoque. N'ayant pu trouver de solution mathématique élégante nous avons dû résoudre ce problème en mettant au point un algorithme qui choisit la solution la meilleure en fonction de critères de proximité.

La modélisation des rotations par des quaternions nous permet d'appliquer les méthodes d'interpolation que nous avons définies; de nouvelles techniques de calculs sont à mettre au point si l'on veut exploiter pleinement toutes les propiétés de l'algèbre des quaternions.

Les quaternions pourraient remplacer les matrices de rotation dans toutes les étapes des calculs des transformations géométriques appliquées aux objets (certains travaux allant dans ce sens [Lafon 1975, Le Borgne 1987, Shoemake 1985, Yahia et Gagalowicz, 1989 et permettre, si la réalisation cablée est possible, des opérations complexes en temps réel.

7.3 Programmation du graphisme interactif

Bien que fortement dépendante des standards et des produits proposés par les constructeurs d'ordinateurs, la programmation des stations graphiques permettant l'animation graphique interactive mérite d'être étudiée sérieusement. On pourra toujours mettre au point n'importe quelle application en faisant tourner des programmes écrit dans un langage classique, faisant appel aux fonctions graphiques de la librairie propre à la machine, gérant l'interactivité par des moyens classiques de programmation *temps réel*, le graphisme interactif étant ainsi traité comme une application parmi d'autres. Il serait préférable de considérer l'animation graphique interactive comme un but pour lequel devront être créés des outils qui restent pour la plupart encore à définir. Le fait de construire une structure de données graphique, mise à jour par un réseau de fonctions **cadencé par les données**, peut être une **réponse** à ce problème. Les réseaux cadencés par les données permettent d'exprimer simplement le **parallélisme** qui est inhérent aux applications temps réel, qui, lorsque les machines

multiprocesseurs seront enfin disponibles, pourra être **effectivement réalisé**.

7.4 Cadencement par les données

Le cadencement par les données pose de sérieux problèmes lorsque l'on veut réaliser des réseaux performants. Cela n'est pas spécifique au cadencement par les données mais provient, en grande partie, de la difficulté d'établir des algorithmes parallèles efficaces.

Les problèmes qui apparaissent à la programmation du cadencement par les données se retrouvent généralement dans d'autres types de langages. Si l'on n'a pas la possibilité de réaliser des **réseaux récursifs** (ce qui suppose une machinerie compliquée permettant l'empilement des données et la gestion des appels qui est dans une certaine mesure contraire à la notion de cadencement) on perd évidemment la facilité offerte par l'écriture fonctionnelle qui nous est maintenant familière.

Cette notation fonctionnelle n'est pas encore vraiment adaptée à l'expression de fonctions opérant sur des suites de données devant être traitées en **pipeline**. C'est parce que les réseaux de fonctions cadencés par les données doivent pouvoir gérer cela qu'ils paraissent compliqués.

Le **parallélisme** et le **temps réel** sont d'autres sources de difficultés, liées aux problèmes de synchronisation et de contraintes de temps de réponse que cela suppose. Le cadencement par les données semble, là, être une solution naturelle.

Beaucoup reste à faire.

Annexe A

La station graphique PS300 Evans & Sutherland

A.1 Avertissement

Généralement une application de graphisme interactif est fortement dépendante de l'ordinateur sur lequel elle est implantée. Il est difficile, à l'heure actuelle, de dissocier nettement les aspect logiciels et matériels; bien que certains standards graphiques tendent à s'imposer (PHIGS semble être en bonne voie), cela signifie encore souvent perte d'efficacité.

Les stations graphiques *PS300* Evans & Sutherland sont à cet égard peu compatibles avec les autres machines du marché. Leurs performances sont remarquables (ceci explique peut être cela), surtout si l'on tient compte du fait qu'elles existent maintenant depuis bientôt dix ans.

Ce chapitre est une présentation presque complète de la station *PS300* Evans & Sutherland, il est inspiré des manuels [PS300 *Users Manual*].

A.2 Introduction

Le *PS300* Evans & Sutherland est une station graphique **3D** avec écran à **balayage cavalier** [Foley 1987].

Il permet l'affichage d'objets graphiques complexes, composés de plus de

50000 vecteurs manipulables directement par l'utilisateur. Ses performances autorisent l'**animation en temps réel** à l'écran.

La description des objets graphiques tridimensionnels se fait de manière **hiérarchisée et structurée** grâce à un langage de programmation évolué.

L'utilisateur peut agir sur les objets visibles à l'écran par l'intermédiaire des périphériques: boutons, tablette graphique, manette. Les programmes gérant cette interactivité sont dits **cadencés par les données** (Data Driven).

A.3 Principe de fonctionnement

La structure de données graphique, créée par l'utilisateur sur l'ordinateur hôte, est implantée en mémoire RAM par le processeur graphique GCP *Graphic Control Processor)*. Elle peut être modifiée à tout moment en temps réel par celui-ci.

A chaque cycle de rafraîchissement, 60 fois par seconde, l'image à afficher à l'écran est entièrement reconstruite à partir de la structure de données graphique.

L'ACP *(Arithmetic Control Processor)* balaye les arborescences de la structure de données pour en extraire les informations nécessaires à la création de l'image. Il effectue également les produits de matrices correspondant aux transformations géométriques à appliquer aux différents objets graphiques.

Le PLS *(PipeLine Subsystem)* traite de manière asynchrone les données fournies par l'ACP, il effectue essentiellement les opérations de clipping, de calcul de perspective et de fenêtre d'affichage.

Les ordres de tracé ainsi créés sont convertis en commandes analogiques par le LGS *(Line Generator Subsystem)*. Ces commandes, déflection et intensité de faisceau, sont envoyées à l'écran cathodique.

A.4 Description des différents modules

A.4.1 *Mass Memory*

La mémoire centrale du *PS300*, d'une capacité de 1 à 4 mégaoctets, contient la structure de données graphique et les programmes du processeur graphique GCP.

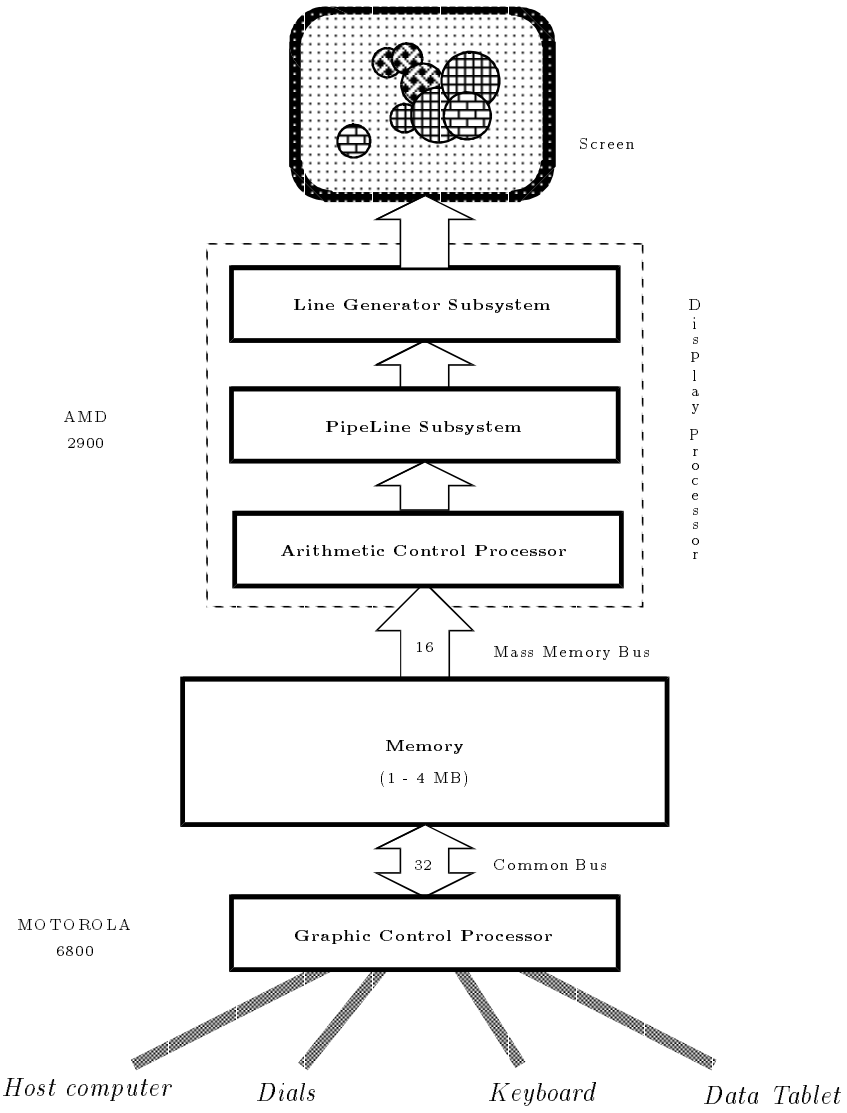


Figure A.1: Schéma général de l'architecture du PS300, d'après [PS300 *Users Manual*].

C'est une mémoire MOS à double accès par l'intermédiaire des bus *Mass Memory Bus* (32 bits) et *Common Bus* (16 bits).

A.4.2 GCP *Graphic Control Processor*

Le processeur graphique est un microprocesseur *MOTOROLA 68000*. Il communique avec la mémoire à travers le bus rapide.

Il est connecté aux périphériques:

- disquette 5"
- ordinateur hôte
- périphériques interactifs:
  - clavier
  - boutons
  - tablette graphique
  - manette (en option)

Il gère :

- la mémoire
- la structure de données graphique
- les communications avec l'ordinateur hôte
- les périphériques interactifs par l'intermédiaire du réseau cadencé par les données créé par l'utilisateur.

A.4.3 *Floppy Disk Drive*

L'unité de disquette 5" n'est généralement utilisée que pour l'initialisation du système (bootstrap). Elle peut néanmoins contenir une structure graphique, rendant ainsi le *PS300* entièrement indépendant de l'ordinateur hôte.

A.4.4 ACP *Arithmetic Control Processeur*

Ce premier étage du processeur d'affichage *(Display Processor)* est constitué d'un microprocesseur en tranche *(AMD2900)*, d'une logique de commande et de 8 processeurs 4 bits spécialement dédiés aux opérations de multiplications. Ces processeurs très rapides permettent de réaliser en un soixantieme de seconde toutes les opérations nécessaires au calcul des transformations géométriques à appliquer à l'ensemble des points à tracer.

Puissance de la machine

En supposant qu'il y ait 40000 vecteurs à tracer, il faudra effectuer 640000 multiplications (16 multiplications étant nécessaires pour calculer la transformation à appliquer à un point), 60 fois par seconde. Tous le calculs se font en arithmétique flottante, la puissance serait donc de 38.4 MFlops. Remarquons que le *PS300* a été la première machine graphique à proposer une arithmétique flottante [Foley 1987].

Principe (présumé) de fonctionnement de l'ACP

Les informations contenues dans la stucture de données graphique, sont organisées en arborescence. A chaque noeud de l'arbre correspond une transformation géométrique (rotation, translation, homothétie, coloration, etc.) applicable à un nombre quelconque de sous-branches qui elles-mêmes sont à nouveau de noeuds ou, en feuille, des listes de vecteurs.

L'ACP parcourt cette arborescence en ordre *préfixé*. Commençant par la racine de l'arbre, il traite chacune des branches, jusqu'aux feuilles, puis passe à la branche immédiatement adjacente, puis à la suivante...Les branches pour lesquelles il n'existe pas d'ordre explicite d'affichage ne sont pas explorées.

Lors du parcours d'une branche, à chaque noeud, il effectue la composition des transformations géométriques, mémorisant ainsi dans une matrice le chemin parcouru depuis la racine. Les bouts de branches, c'est à dire les feuilles, sont des listes de coordonnées correspondant à des ordres de tracé. L'ACP n'a plus qu'à multiplier ces suites de points par cette matrice. Toutes ces opérations sont faites sur des

vecteurs en coordonnées homogènes à 4 dimensions, permettant ainsi les projections en perspective.

Lorsque l'arborescence ou le nombre de vecteurs deviennent trop grands, il est possible que l'ACP n'ait plus le temps de traiter toute la structure graphique en un soixantième de seconde. L'affichage se fait alors plus lentement, l'image semble papilloter.

A.4.5 PLS *PipeLine Subsytem*

Les données correspondant aux coordonnées exprimées dans "l'espace utilisateur" fournies par l'ACP sont transformées en coordonnées "écran" par le PLS. C'est un processeur asynchrone pipeline dont la fonction essentielle est de réaliser les opérations de projections perpectives ou parallèles, de clipping tridimensionnel et d'affichage.

A.4.6 LGS *Line Generator Subsytem*

Les commandes réelles de tracé ainsi créées par le PLS sont converties en signaux analogiques par le LGS. Il calcule la vitesse de tracé, élimine les vecteurs courts, c'est-à-dire ceux dont la longueur ne dépasse pas 1mm. Le LGS détermine également quel vecteur a été désigné par l'utilisateur au moyen de la tablette graphique: le système connait en permanence la position du stylo sur celle-ci, et peut donc savoir, au moment du *clic*, quel segment tracé à l'écran *passait* par cette position. Il crée les commandes analogiques de déflection et d'intensité de faisceau qui sont transmises à l'écran cathodique.

A.4.7 HCP *HardCoPy*

Une copie de l'écran peut être envoyée à une imprimante du type Versatec 80 ou Hewlett Packard par l'intermédiaire d'un port parallèle 8 bits. Notons également qu'il est possible de renvoyer à l'ordinateur hôte toute l'image affichée à l'écran. Cette "SoftCopy" permet de créer un fichier format PostScript qu'il suffit d'imprimer, les figures 6.1, **??** de cette thèse ont ainsi été créées.

A.5 La structure de données graphique

A.5.1 Introduction

La structure de données graphique est implantée en mémoire par le processeur graphique GCP (Motorola 68000). L'utilisateur décrit cette structure au moyen d'un langage hierarchisé et structuré.

Nous n'aborderons en fait que ce langage de programmation, ne disposant en effet que de peu d'informations concernant l'implantation réelle de la structure de données en mémoire.

A.5.2 Pourquoi une structure de données graphique?

La fonction essentielle du *PS300* Evans & Sutherland est de représenter sur l'écran des "objets" tridimensionnels définis par l'utilisateur. Celui-ci n'a pas à "créer" la représentation mais à definir d'une manière simple les objets tels qu'ils sont, tels qu'ils évoluent dans leur espace propre. Ils peuvent être décrits de manière logique, en respectant leur structure,

l'utilisateur **MODELISE** ses objets.

Il faut également définir de quelle manière les "scènes" ainsi créées doivent être représentées à l'écran, quelles parties de l'espace doivent être "vues", quelle projection adopter,

l'utilisateur **VISUALISE** son espace.

La structure de données ainsi créée n'est pas figée. Les paramètres la définissant peuvent être modifiés de manière interactive par l'utilisateur à travers le programme cadencé par les données (Data Driven),

l'utilisateur **AGIT** sur la structure.

A.5.3 Description et manipulation de la structure de données

Le langage de programmation PSCL (PS300 Command Language) permet une description aisée des aspects "modélisation" et "visualisation" de la structure graphique. Le langage de programmation PSDDNL (PS300 Data Driven Network Language) est l'outil de définition des "réseaux cadencés par les données". Ces langages sont décrits dans ce qui suit et au chapitre 3.

A.6 Le langage PSCL

A.6.1 Principes généraux

Le but d'un "programme" écrit dans ce langage est de:

- décrire des objets tridimensionnels dans leur espace,
- définir la manière de les afficher à l'écran.

A chaque objet on associe:

- sa forme (ou plus précisément sa "structure"),
- sa position et son orientation dans l'espace,
- ses attributs (couleur, éclairage, visibilité etc.).

Un observateur "regarde" les objets dans leur espace, l'image obtenue à l'écran est définie par:

- la position de l'observateur,
- la ligne de vue,
- le type de projection utilisée.

A.6.2 Le système de coordonnées

Les objets sont décrits dans un espace tridimensionnel défini par un repère orthonormé XYZ représenté figure A.2.

Par convention le repère est **gauche** c'est-à-dire que si X est orienté vers la droite, Y vers le haut, alors l'axe des Z pointe vers l'arrière de l'écran.

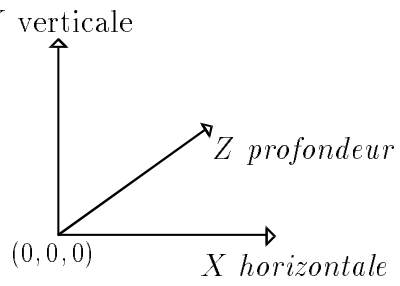


Figure A.2: Repère gauche $Oxyz$. Z pointe vers l'arrière.

A.6.3 Les primitives graphiques

Tout objet graphique, aussi complexe soit-il, est défini à partir d'objets élémentaires (primitives) qui sont des points ou des segments de droite.

Ces objets élémentaires sont appelés **vector_list**.

Ils sont décrits de la manière suivante: pour le cube fait de segments,

```
CUBE:= vector_list itemized
        p 0, 0, 0
        l 2, 0, 0       l 2, 2, 0
        l 0, 2, 0       l 0, 0, 0
        l 0, 0, 2
        l 2, 0, 2       l 2, 2, 2
        l 0, 2, 2       l 0, 0, 2
        p 0, 0, 0       l 0, 0, 2
        p 2, 2, 0       l 2, 2, 2
        p 0, 2, 0       l 0, 2, 2
        p 2, 0, 0       l 2, 0, 2 ;
```

ou les points de chaque face,

```
UDTQCS:= vector_list dots
          p 1.0, 1.0, 0.0
          p 0.5, 2.0, 0.5
          p 0.5, 2.0, 1.5
```

p 1.5, 2.0, 1.5
p 1.5, 2.0, 0.5
p 2.0, 1.0, 1.0
p 2.0, 0.5, 0.5
p 2.0, 0.5, 1.5
p 2.0, 1.5, 1.5
p 2.0, 1.5, 0.5 ... ;

p signifie **Position**, c'est à dire qu'il faudra se positionner sans tracer au point défini par ses coordonnées x, y, z.

l signifie **Line**, c'est à dire qu'il faudra tracer un segment de droite partant du point défini précédemment jusqu'au point de coordonnées x, y, z.

Dans une instruction **dots** (points) on ne trace pas les segments mais uniquement les points.

A.6.4 Les transformations géométriques

Des objets plus complexes peuvent maintenant être construits à partir de ces "primitives".

- Ainsi on pourra créer l'objet `DE`, composé des deux objets `CUBE` et `UDTQCS`, par

```
DE:=instance of CUBE,UDTQCS ;
```

- A tout objet graphique on peut appliquer des transformations géométriques, c'est-à-dire des translations, des rotations, et des changements d'échelle, ou d'une manière générale toute matrice 3×3.

  Par exemple :

```
GROSDE := scale by 3 applied to DE ;
ROTDE  := rotate in y 60 applied to DE ;
RORO   := rotate in x 20 applied to ROTDE ;
VOLDE  := translate by 1,2,0 applied to RORO ;
```

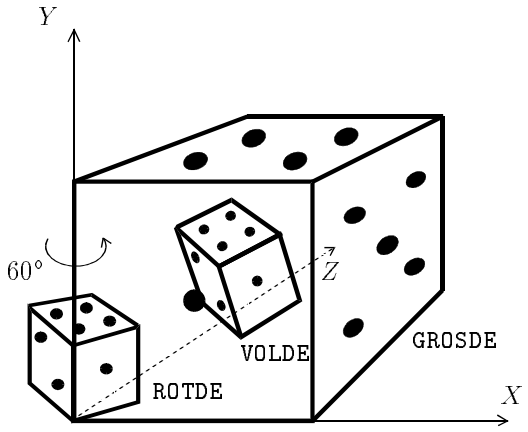


Figure A.3: Les objets GROSDE, ROTDE et VOLDE.

correspond à l'image de la figure A.3.

Nous connaissons maintenant l'essentiel pour pouvoir construire un objet.

A.6.5 Les structures

Dans tout ce que nous venons de voir, nous avons toujours nommé chaque "nouvel" objet créé par une commande. En effet, et c'est important, chaque commande crée un nouvel objet sans pour autant détruire celui auquel s'applique cette commande. Ainsi l'objet DE peut être utilisé en plusieurs endroits. L'ensemble des objets disponibles peut se schématiser comme sur la figure A.4. On peut également créer d'autres objets à partir de RORO, de CUBE,de GROSDE, etc..

Il arrive fréquemment qu'on ait besoin d'appliquer plusieurs transformations à un même objet sans pour autant se servir ailleurs des objets intermédiaires, dans ce cas, essentiellement pour des raisons de facilité d'écriture et de compréhension, il est possible de regrouper toutes ces opérations dans une même structure.

Par exemple,

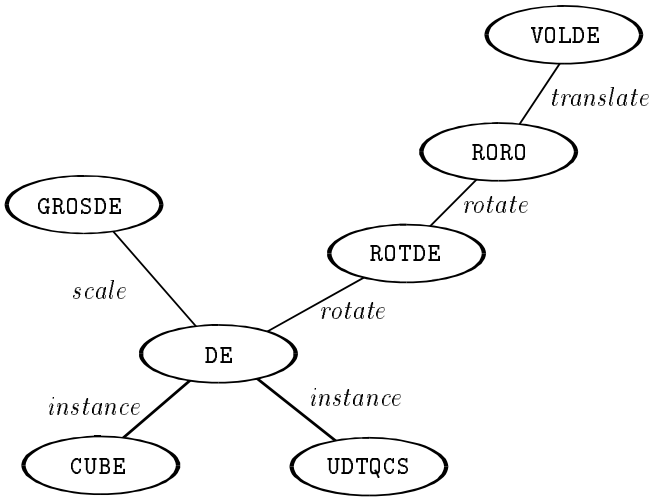


Figure A.4: Construction hiérarchique de `GRODE` et `VOLDE` à partir des listes de vecteurs `CUBE` et `UDTQCS`.

```
VOLDE:=begin_structure
                translate by 1,2,0 ;                (1)
                rotate in x 20 ;                    (2)
                rotate in y 30 applied to DE ;      (3)
        end_structure ;
```

est équivalent à

```
VOLDE :=translate by 1,2,0 applied to RORO ;
RORO  :=rotate in x 20 applied to ROTDE ;
ROTDE :=rotate in y 30 applied to DE ;
```

... et évite les sempiternels `applied to` et de nommer tous les objets.

Toute commande de la ligne (i) s'applique implicitement à l'objet défini par la ligne $(i+1)$, s'il n'y a pas d'ordre `applied to` explicite.

Dans une même structure peuvent figurer plusieurs objets :

```
DESDES:=begin_structure
            translate by 1,2,0 ;                    (1)
            rotate in x 20 ;                        (2)
            rotate in y 30 applied to DE ;          (3)
            scale by 3 applied to DE ;              (4)
        end_structure ;
```

Remarquons que la rotation (3) ne s'applique pas à la ligne (4) puisqu'elle s'applique à l'objet `DE`; l'objet `DESDES` est de ce fait composé de deux dés.

Chaque ligne d'une structure peut être nommée :

```
DESDES:=begin_structure
            TRA  :=translate by 1,2,0 ;             (1)
            RYX  :=rotate in x 20 ;                 (2)
            RY   :=rotate in y 30 applied to DE ;   (3)
            SC   :=scale by 3 applied to DE ;       (4)
        end_structure ;
```

Chaque objet défini à l'intérieur d'une telle structure peut être référencé par :

`DESDES.TRA`, `DESDES.RY`, etc..

Les structures peuvent être imbriquées les unes dans les autres en parenthésant par `begin_structure` et `end_structure`:

```
JULES:=begin_structure
                ...
                TOTO:=begin_structure
                                ...
                                OLAF:=begin_structure
                                            ...
                                        end_structure ;
                                ...
                                ...
                                LULU:=begin_structure
                                                ...
                                        end_structure ;
                                ...
                        end_structure ;
                ...
        end_structure ;
```

A.6.6 L'attribut couleur

Nous savons créer des primitives et leur appliquer des transformations géométriques pour construire des objets plus complexes.

A tout objet on peut associer une couleur définie par

- une valeur de couleur de 0 à 360 (un tour):
  0 bleu, 120 rouge, 240 jaune, 360 bleu.
- une valeur de saturation de 0 à 1 :
  0 non saturé (blanc), 1 saturé.

La commande de coloriage est :

```
COLDE:=set color 120.0 , 0.75 applied to GROSDE ;
```

où `120.0` correspond à la couleur rouge, `0.75` à la saturation.

La couleur d'un objet X ne peut pas être redéfinie par une commande de coloriage d'un objet Y qui fait référence à X.

Exemples :

```
X := set color 120.0 , 0.75 applied to OBJ1 ;
E := instance of X,OBJ2 ;
Y := set color 240,1 applied to E ;
```

`OBJ1` et `OBJ2` ne sont pas coloriés, `X` est l'objet `OBJ1` colorié en rouge, `E` est composé de `OBJ1` rouge et de `OBJ2` sans couleur, `Y` sera composé de `OBJ1` rouge (et non pas jaune) et de `OBJ2` jaune.

A.6.7 Les références conditionnelles

Comme nous le verrons plus loin la structure de données définie "statiquement" par les commandes du langage de description PSCL pourra évoluer en fonction des ordres donnés par les périphériques d'interaction. Ainsi, on voudra, par exemple, ne plus afficher tel objet, appliquer telle transformation à une autre structure. Ces **références conditionnelles** doivent être prévues lors de la description statique de la structure de données.

A toute branche de l'arborescence peuvent être associés :

- 1 variable entière `level_of_detail`
- 15 variables logiques `conditional_bit`

La valeur des entiers `level_of_detail` est comprise entre 0 et 32764.

Les 15 variables logiques `conditional_bit1, ... conditional_bit15` peuvent prendre les valeurs "vrai" ou "faux".

Ces valeurs sont définies à priori par les commandes PSCL mais peuvent être modifiées par le réseau DDN.

Exemple d'utilisation de `level_of_detail`

Soit à définir la structure correspondant à un "film" composé de 10 images. Chaque image est définie par un objet spécifié par ailleurs (par exemple 10 vector_lists différentes `IMAGE0, IMAGE1, ... IMAGE9`).

Définissons FILM par :

```
FILM:=begin_structure
        set level_of_detail to 0 ;
        if level_of_detail = 0 then IMAGE0 ;
        if level_of_detail = 1 then IMAGE1 ;
        if level_of_detail = 2 then IMAGE2 ;
        if level_of_detail = 3 then IMAGE3 ;
        if level_of_detail = 4 then IMAGE4 ;
        if level_of_detail = 5 then IMAGE5 ;
        if level_of_detail = 6 then IMAGE6 ;
        if level_of_detail = 7 then IMAGE7 ;
        if level_of_detail = 8 then IMAGE8 ;
        if level_of_detail = 9 then IMAGE9 ;
      end_structure ;
```

Tel quel, l'objet FILM est en fait l'objet IMAGE0. Si, par l'intermédiaire d'un réseau DDN (voir plus loin), on affecte la valeur 1 au "level_of_detail", FILM sera l'objet IMAGE1.

Puis 2, 3 etc., le film contitué des images IMAGE0, IMAGE1, ... IMAGE9 se "déroule" à l'écran.

Exemple d'utilisation de `conditional_bit`

```
TOTO:=begin_structure
        set conditional_bit 6 on ;
        if conditional_bit 6 is on then RIRE ;
        if conditional_bit 6 is off then PLEUR ;
        set conditional_bit 3 off ;
        if conditional_bit 3 is on then CHAPEAU ;
        instance of CORPS ;
      end_structure ;
```

`TOTO` a ainsi un `CORPS`, une tête `RIRE` et pas de `CHAPEAU`.

En modifiant `conditional_bit6` on peut faire pleurer `TOTO`, en modifiant `conditional_bit3` on lui offre un `CHAPEAU`.

Remarquons que le fait d'avoir un chapeau n'a à priori aucune influence sur l'humeur de `TOTO`, les différents `conditional_bit` étant indépendants.

A.6.8 Récapitulons...

Nous savons créer des objets élementaires

Vector_List

nous savons définir de nouveaux objets en appliquant des transformations géométriques

Instance, Translate, Rotate, Scale

nous pouvons colorier

Set Color

enfin, nous pouvons nous servir de références conditionnelles par

level_of_detail

et

conditional_bit.

Toutes ces opérations doivent être décrites de manière intelligente, en structurant et en hiérarchisant afin de correspondre le plus possible à la "réalité".

Ainsi la description d'un robot doit être faite de manière à ce que les mouvements du membre supérieur s'appliquent correctement au bras, à l'avant-bras, à la main, que les déplacements du robot s'appliquent au corps, au bras ... et donc à l'avant-bras et à la main.

A.6.9 Visualisation

La scène est créée, les objets existent,... dans un espace à trois dimensions. Il faut maintenant les visualiser sur l'écran, en deux dimensions.

Pour cela, plaçons un observateur dans l'espace tridimensionnel des objets, (en fixant sa **position** dans le répère XYZ), définissons l'endroit vers où il regarde (sa **ligne de vue**), comment il regarde (type de **projection** et **éclairage**), quels objets ou parties d'objets il doit voir (**plans de coupe**).

Ligne de vue

La position de l'observateur dans l'espace objet est donnée par :

```
LOOK FROM x,y,z
```

La direction vers où il regarde est donnée par :

```
LOOK AT a,b,c
```

L'orientation de la tête de l'observateur, c'est-à-dire "sa" verticale, est donnée par :

```
LOOK UP u,v,w
```

Ces trois commandes peuvent s'écrire en une :

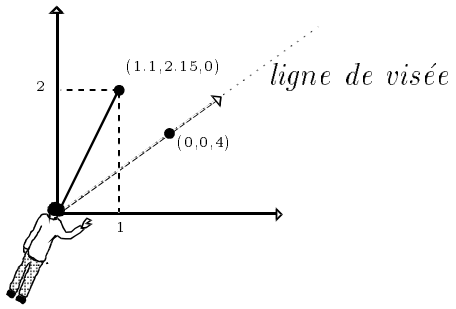


Figure A.5: "`LOOK FROM 0, 0, 0 AT 0, 0, 4 UP 1.1, 2.15, 0`". L'observateur, dont l'oeil est situé à l'origine $(0,0,0)$, regarde en direction des Z positifs $(0,0,4)$ (vers l'arrière de l'écran) et est tourné de telle sorte que l'axe de son corps passe par le point $(1.1, 2.15, 0)$.

```
LOOK FROM x,y,z AT a,b,c UP u,v,w
```

Par exemple, pour indiquer que l'observateur située à l'origine regardant en direction des Z croissant avec la tête inclinée de telle sorte que son axe passe par le point $(1.1, 2.15, 0.0)$, l'instruction sera `LOOK FROM 0,0,0 AT 0,0,1 UP 1.1, 2.15, 0.0` [fig.A.5].

Projection

L'image fournie par un appareil photographique dépend de la position et de l'orientation de l'appareil, mais aussi du réglage de l'objectif, essentiellement de la focale (zoom), de la position du plan de focalisation (mise au point) et de l'ouverture du diaphragme (profondeur de champ). Les objets sont vus avec plus ou moins de netteté suivant qu'ils sont plus ou moins proches du plan de focalisation (mise au point et profondeur de champ).

Contrairement à l'image obtenue par photographie, les objets situés loin du plan de focalisation ne seront pas flous mais pourront éventuellement ne pas être affichés (Depth Clipping), c'est pour cette raison qu'il suffit de définir les plans avant et arrière et non pas la position du plan de focalisation et la profondeur de champ.

Tous ces paramètres sont exprimés à travers la commande

```
FIELD_OF_VIEW
        ANGLE a
        FRONT_BOUNDARY=f
        BACK_BOUNDARY=b ;
```

Un objectif à décentrement permet de voir une région de l'espace décentrée par rapport à la ligne de mire.

Cela s'exprime par la commande

```
EYE_BACK
     z BACK
     l LEFT (ou r RIGHT)
     u UP (ou d DOWN) FROM SCREEN AREA
     w WIDE
     FRONT_BOUNDARY=f
     BACK_BOUNDARY=b ;
```

Signalons que ces deux transformations peuvent s'écrire sous la forme de matrices 4×4 [Foley 1987]; l'utilisateur peut créer sa propre matrice par la commande

```
MATRIX_4x4
        m11,m12,m13,m14
        m21,m22,m23,m24
        m31,m32,m33,m34
        m41,m42,m43,m44 ;
```

Un objectif d'appareil photographique (ainsi que les commandes `Field_of_View` et `Eye_back`) donne toujours une vue en perspective, et donc une représentation "déformée" des objets (ligne de fuite, parallèles qui se rejoignent, etc.) [Foley 1987].

La projection "orthographique" qui est une projection parallèle à la ligne de mire est définie par la commande

```
WINDOW
    X=xmin:xmax
    Y=ymin:ymax
    FRONT_BOUNDARY=f
    BACK_BOUNDARY=b ;
```

Eclairage

Nous venons de définir la position de l'observateur

```
LOOK FROM AT UP
```

et le type de projection

```
FIELD_OF_VIEW, EYE_BACK ou WINDOW
```

il reste à définir l'**éclairage** . Celui-ci peut être plus ou moins intense, plus ou moins contrasté.

Les commandes `Field_of_View`, `Eye_back` et `Window` spécifient les plans avant et arrière. La **luminosité** d'une scène est définie de telle sorte qu'elle varie continuellement du plus intense (plan avant) au moins intense (plan arrière). Les zones éloignées (en profondeur) sont plus sombres. Les valeurs sont toujours comprises entre 0 (on ne voit rien) et 1 (le plus clair possible).

Ainsi la commande

```
SET INTENSITY ON 0.1:0.8
```

affecte la luminosité 0.8 au plan avant et 0.1 au plan arrière. Un objet situé à égale distance des plans aura une intensité de $(0.8 + 0.1)/2 = 0.45$.

Le contraste ainsi appliqué à la scène peut être atténué par la commande

```
SET CONTRAST c
```

Si `c` a la valeur 1 (contraste maximum) la commande n'a aucun effet. Si `c` a la valeur 0 (pas de contraste) l'intensite minimale (ici 0.1) est remplacée par l'intensite maximale (0.8), il n'y aura pas de variation de luminosité en fonction de la profondeur.

Les valeurs intermédiaires de `c` atténuent ainsi plus ou moins cette valeur d'intensité minimale définie par la commande `set intensity`. (quatre valeurs seulement sont prises en compte, $0, \frac{1}{3}, \frac{2}{3}$ et 1)

Remarque, la commande d'intensité s'écrit en fait

```
SET INTENSITY o min:max
```

où `o` peut prendre la valeur "on" ou "off". `set intensity off` signifiant qu'il ne faut pas tenir compte des valeurs "min" et "max" mais leur affecter les valeurs 0 et 1 (pas de contraste).

Cette technique de luminosité décroissante en Z donne une très bonne impression de profondeur à l'image bidimensionnelle.

Les objets ou parties d'objets situés en deçà du plan avant ou au delà du plan arrière peuvent ne pas être affichés grâce à la commande

```
SET DEPTH_CLIPPING ON
```

La commande

```
SET DEPTH_CLIPPING OFF
```

permettra l'affichage de ces zones (avec l'intensité min ou max correspondante). L'image ainsi créée est affichée à l'écran.

La commande

```
VIEWPORT
      HORIZONTALE = Hmin:Hmax
      VERTICALE = Vmin:Vmax
      INTENSITY = Imin:Imax ;
```

indique que l'image doit être affichée dans la partie d'écran définie par bord gauche en `Hmin`, droit en `Hmax`, bord inférieur en `Vmin`, supérieur en `Vmax` L'écran complet est définie par `(Hmin:Hmax)=(-1:1)`, `(Vmin:Vmax)=(-1:1)`;

Le centre de l'écran est en $(0,0)$. Remarquons que si la fenêtre n'est pas carrée (`Hmax-Hmin` $\neq$ `Vmax-Vmin`), l'image est déformée (à moins de définir dans la commande `window` les valeurs adéquates de X et Y).

Récapitulons ...

Un observateur défini par

```
LOOK FROM f AT a UP u
```

regarde (d'une certaine manière)

```
FIELD_OF_VIEW, EYE_BACK ou WINDOW
```

une scène composée d'un ensemble d'objets décrits dans leur espace.

Cette scène éclairée par

```
SET INTENSITY, SET CONTRAST
```

peut être vue partiellement grâce à

```
SET DEPTH_CLIPPING
```

L'image vue par l'observateur est affichée dans la partie d'écran définie par

```
VIEWPORT
```

A.6.10 Hiérarchie

L'image affichée sur telle ou telle **partie de l'écran** est ce qui est **vu d'une certaine manière** par un observateur, **placé** en un point de l'espace tridimensionnel, **regardant** dans une direction donnée, une scène composée d'objets graphiques structurés de façon hiérarchique par des opérations d'instanciation, de transformations géométriques simples et d'affectations d'attributs de couleur et luminosité.

```
display bidule ;

bidule:=begin_structure
          viewport ...;
          set contrast ...;
          field_of_view[1]...;
          look from ...at ...up ...;
          translate ...;
          rotate ...;
          scale ...;
          set color ...;
          set intensity ...;
        end_structure ;
```

Ici l'arborescence n'apparaît pas, de chaque nœud ne part qu'une seule branche. L'utilisation des commandes `begin_structure - end_structure`, `instance of` et `applied to` permet toutes les constructions.

L'ordre des commandes n'est pas nécessairement celui indiqué ci-dessus. Il faut néanmoins respecter la hiérarchie induite par les commandes de constructions géométriques, de visualisation et de fenêtre d'affichage.

Il faut respecter la hiérarchie

[1] ou `eye_back` ou `window`

```
display bidule ;

bidule:=begin_structure
        fenetre d'affichage
        mode de visualisation
        definition d'objets
    end_structure ;
```

A.7 Le Langage PSDDNL

A.7.1 Principes

Les différents paramètres d'une structure de données graphique définie par le langage de description PSCL peuvent être modifiés **interactivement** par l'utilisateur. Celui-ci peut **agir** sur les objets graphiques par l'intermédiaire des boutons, du clavier, de la tablette graphique ou même à travers un programme implanté sur l'ordinateur hôte: à tel bouton sera associé la rotation autour de l'axe des x, à tel autre un déplacement en z.

Cette interactivité est gérée par un réseau de fonctions **cadencé par les données** qui doit être créé par l'utilisateur. Dans le chapitre 3 nous avons introduit cette notion de *cadencement par les données*, nous ne présentons ici que les grandes lignes du langage de programmation PSDDNL (*PS300 Data Driven Network Langage*) et son utilisation pour le traitement de l'interactivité.

Un exemple: la translation

Considérons la commande suivante

```
VOLDE:=translate by 1.05 , 2.4 , 0 applied to RORO ;
```

L'objet `VOLDE` est obtenu par translation de l'objet `RORO`. Affectons, par connection de la boite système `f:dials`, au bouton 1 la fonction de translation dans la direction X et au bouton 2 la translation en Y.

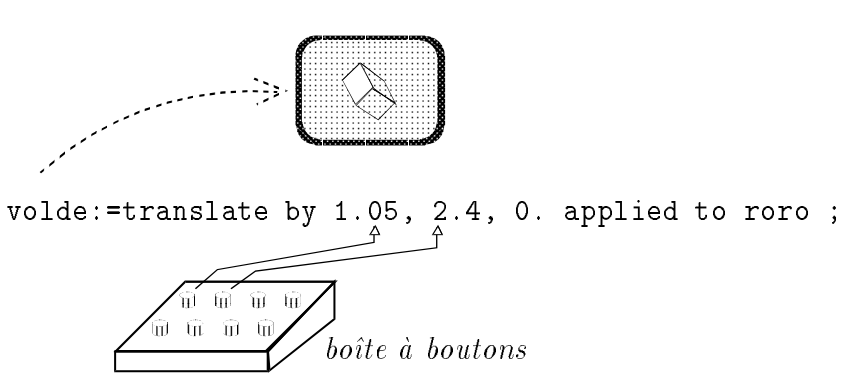


Figure A.6: Les boutons 1 et 2 permettent de déplacer l'objet. Les boutons activent le réseau de translation de la figure A.7 qui met à jour l'instruction "`volde:= ...`" dans la structure de données.

Lorsque l'utilisateur tourne le bouton 1, celui-ci (la boîte `f:dials`) "émet" une valeur que nous devons interprèter comme étant un déplacement relatif en X. Cette valeur du déplacement devra être additionnée à la valeur absolue précédemment enregistrée. Les déplacements "émis" par le bouton 2 affecteront de même la position en Y (voir figure A.7).

L'instruction "`translate by 1.05,2.4,0`", implantée dans la structure de données graphique doit être mise à jour. La valeur à modifier est celle du vecteur tridimensionnel associé à la position absolue (x,y,z). Cette mise à jour doit être faite à chaque "émission" de l'un des boutons 1 ou 2. Ceci est fait par le réseau de la figure A.7.

La valeur scalaire dX correpondant au déplacement relatif en X est transformée en un vecteur 3D $vX = (dx, 0, 0)$ par la boîte (noire?) nommée ici "`CONV_dX_vX`". De même le scalaire dY est converti en vecteur 3D $vY = (0, dY, 0)$ par `CONV_dY_vY`.

Ces vecteurs sont additionnés à la position absolue $vXYZ$ (vecteur 3D), qui est "stockée" dans la boîte `POS_ABS`. Toute valeur émise par l'un des boutons "active" la boîte de conversion correspondante, cette boîte "émet" à son tour une valeur (vecteur 3D) qui "activera" la boîte d'addition. La nouvelle position calculée par `POS_ABS` est "envoyée" à la structure de données, l'instruction "translate by 1,2,0" est alors modifiée. La connection du réseau sur la structure de données graphique est faite par des instructions du type `connect POS_ABS<1>:<1>volde`.

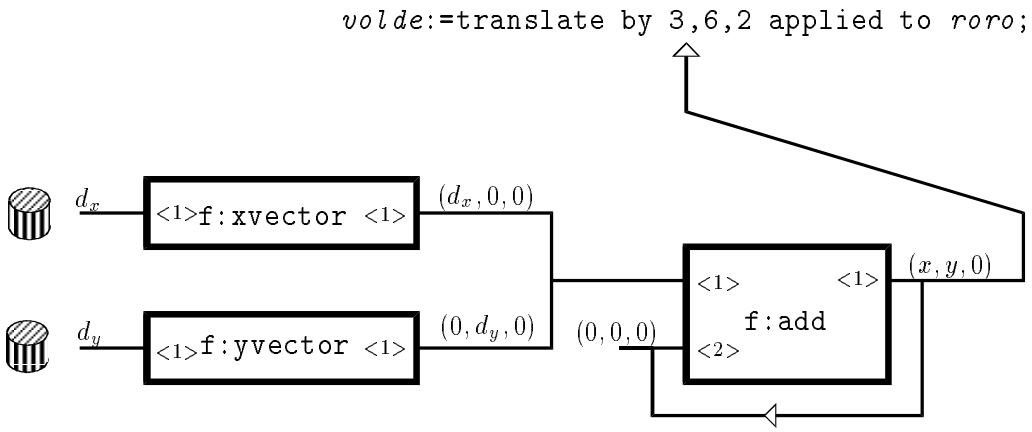


Figure A.7: Translation d'un objet

A.7.2 Le type des données

A chaque donnée circulant sur un arc est associé un type. Les tyes possibles sont:

- scalaires
  - booléen
  - entier
  - flottant
- vecteurs 2D ou 3D d'entiers ou de flottants
- matrices 3×3, 3×4, 4×4,
- chaînes de caractères
- liste de vecteurs produite par la fonction `F:XformData` à partir de la structure de données dont l'image est visible à l'écran.

Ces types ne sont pas spécifiques aux arcs mais sont définis par la fonction et par le type des données en entrée. Ainsi si une fonction `addition` reçoit deux entiers elle émet un entier, si par la suite elle reçoit deux matrices de flottants, elle émet une matrice de flottants.

A.7.3 Les fonctions élémentaires de PSDDNL

Nous donnons ici un aperçu des fonctions élémentaires du langage. Cette liste incomplète devrait néanmoins permettre au lecteur de se faire une idée de ce que la machine *sait faire* et de ce que l'utilisateur doit programmer. Le *PS300* étant une station graphique, les fonctions élémentaires choisies par le constructeur sont orientées dans ce sens. La liste exhaustive de toutes les fonctions existantes (il y en a plus de 100, 200 si l'on tient compte des variantes) se trouve dans [PS300 *Users Manual*].

Opérations arithmétiques et logiques

Les quatre opérations et toutes les comparaisons entre nombres ou valeurs logiques existent, mais aussi entre vecteurs et matrices. Sont également disponibles, le calcul de *sinus, cosinus* (mais pas des fonctions inverses), la racine carrée et des fonctions permettant de calculer des moyennes et des cumuls sur des suites de données.

Manipulation des chaînes de caractères

Le texte affichable à l'écran, sous la forme d'un objet graphique 3D ou par simple message, peut être créé par le réseau lui-même. par les opérations usuelles: concaténations, recherches et extractions de sous-chaînes, calcul de longueur et masques.

Conversion de type

Toutes les conversions sont possibles, entier vers flottant, chaînes de caractères vers entier, etc., mais aussi 2 ou 3 scalaires (entiers ou flottants) vers vecteurs (2D ou 3D), 2 ou 3 vecteurs vers matrice (2×2 ou 3×3), matrice vers vecteurs, vecteur vers scalaires,...

Fonctions de routage

Nous citons principalement la fonction d'aiguillage (`F:Route`) et de sélection (`F:Inputs_Choose`, `F:Boolean_Choose`), la boîte de synchronisation (`F:Sync`) et la fonction de stockage d'une constante (`F:Constant`).

Fonctions géométriques

Elles permettent de créer les matrices associées aux transformations géométriques et aux opérations de visualisation. La fonction `F:Scale` produit la matrice d'homothétie à partir d'un flottant ou d'un vecteur 3D si l'on veut des facteurs d'échelle différents sur les trois axes. On peut créer, à partir d'un flottant, la matrice de rotation autour de l'axe `Ox` (`F:XRotate`, `Oy` (`F:YRotate`) ou `Oz` (`F:ZRotate`).

La definition de l'observateur se fait par les fonctions `F:LookAt` et `F:LookFrom`, les paramètres de projection par `F:Field_of_View`, `F:EyeBack` ou `F:Window`.

Les fonctions horloge

Elles autorisent toutes les possibilités, émission d'une valeur à intervalle de temps régulier, tous les n rafraichissements d'écran, ou création de chien de garde, détection de *timeout*, etc.

Gestion des périphériques

Ce sont des fonctions du système qui permettent de prendre en compte au niveau du réseau les périphériques d'entrée:

- le clavier (`F:Fkeys` et `F:ButtonsIn`),
- la tablette graphique (`F:Tablettin` et `F:Pick`),
- l'ordinateur hôte (`F:Host`),

les périphériques de sortie:

- les afficheurs lumineux (`F:Dlabels` et `F:Flabels`),
- la liaison vers l'ordinateur hôte (`F:HostOut`),
- l'affichage des messages d'erreurs ou d'information (`F:Error`, `F:Message_Display`),
- la recopie d'écran (`F:HardCopy`).

Fonctions spéciales

Remarquons particulièrement la fonction `F:H_Chop` qui est le point d'entrée de l'interpréteur du langage. C'est cette fonction qui reçoit les commandes émises par l'ordinateur hôte mais peut également traiter des chaînes de caractères créées par le réseau, lui permettant ainsi de se modifier lui-même.

A.7.4 Programmation

La machine ne comprend (malheureusement) pas les dessins des réseaux que nous aimerions lui soumettre. Nous devons les lui traduire.

Ainsi le réseau de la figure A.7 se programmera de la manière suivante:

```
CONV_DX_VX:=f:xvector ;                                    (1)
      conn dials<1>:<1>CONVDX_VX ;                         (2)

CONV_DY_VY:=f:xvector ;                                    (3)
      conn dials<2>:<1>CONVDY_VY ;                         (4)

POS_ABS:=f:add ;                                           (5)
      conn CONV_DX_VX<1>:<1> POS_ABS ;                     (6)
      conn CONV_DY_VY<1>:<1> POS_ABS ;                     (7)
      conn                   POS_ABS<1>:<2>POS_ABS ;       (8)
      send V3D(0,0,0) to <2>POS_ABS ;                      (9)
      conn                   POS_ABS<2>:<1>VOLDE ;         (10)
```

Les instructions du langage se résument à:

```
NOM_BOITE:=f:NOMFONCTION ;
conn NOM_BOITE<S>:<E>NOM_BOITE ;
send INFORMATION to <E>NOM_BOITE ;
disc NOM_BOITE<S>:<E>NOM_BOITE ;
disc NOM_BOITE<S>:all ;
init connections ;
```

Une instruction se termine toujours par ";".

La définition d'une connection ne peut pas être faite avant celle de la source; une connection peut être définie avant la destination. La ligne `(1)` doit figurer avant la `(2)`.

Il est possible d'envoyer des valeurs sur n'importe quelle entrée par la commande

`send` *INFO* `to <`*E*`>BOITE`

Ainsi nous avons "initialisé" l'entrée `<2>` de la boite `POS_ABS` par `(9)`.

La commande

`disc BOITE1<`*S*`>:<`*E*`>BOITE2`

déconnecte la sortie `<`*S*`>` de la boite1 de l'entrée `<`*E*`>` de la boite2.

`disc NOM_BOITE0<`*S*`>:all`

signifie que la sortie `<`*S*`>` de nom_boite0 est déconnectée de toutes les entrées auxquelles elle était connectée. La commande `disc all:<`*E*`>NOM_BOITE` n'existe pas, les prédécesseurs d'une entrée n'étant pas connus explicitement.

La commande

`init connections`

supprime toutes les connections.

Remarquons que la sortie `POS_ABS<1>` "reboucle" sur l'entrée `<2>`. De cette façon nous "mémorisons" l'ancienne valeur de la translation dans la file d'attente `<2>POS_ABS`.

Les instructions sont interpretées par le processeur graphique MC68000 au fur et à mesure qu'elles lui arrivent. On peut redéfinir une boîte, supprimer ou rajouter

des connections, initialiser certaines entrées. Les réseaux ainsi définis sont immédiatement actifs, l'information circule entre les boites, la base de donnée graphique est mise à jour; l'image affichée à l'écran évolue en conséquence.

A.7.5 Où créer ces commandes ?

Le processeur graphique est accessible par :

- le clavier
- le lecture de la disquette
- l'ordinateur hôte
- le réseau lui-même.

Le réseau peut créer des instructions à l'aide des fonctions de traitement de chaînes de caractères; ces instructions, envoyées sur l'entrée `<1>` de la boîte appelée `H_CHOP`, sont interpretées par le processeur graphique.

Le réseau peut éventuellement se modifier lui-même.

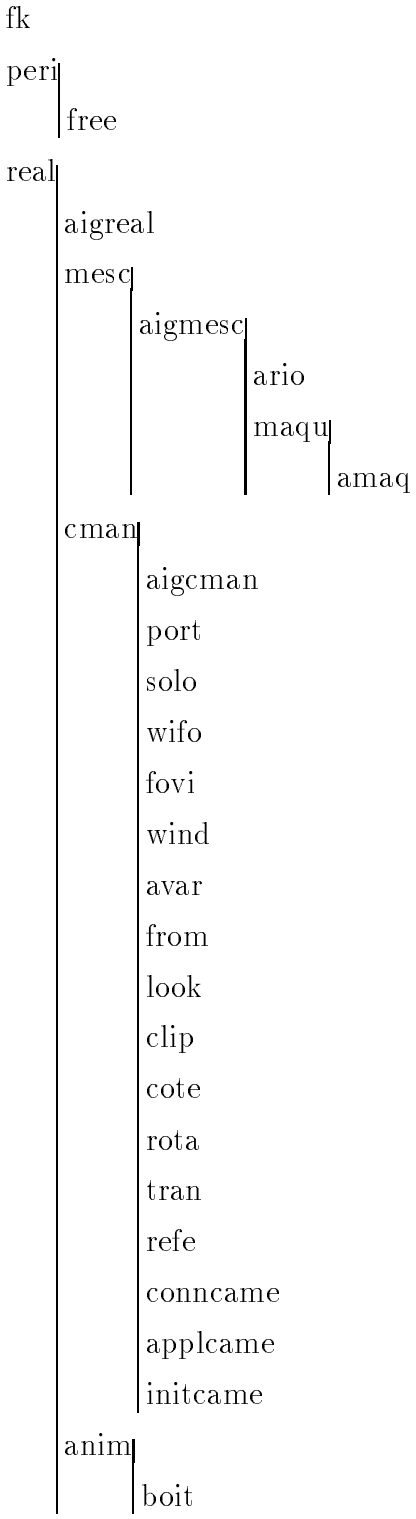
Généralement les instructions sont créées sur l'ordinateur hôte par éditeur de texte ou par programme. Elles sont envoyées au PS300 directement ou par copie de fichiers.

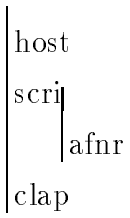
Annexe B

AGRAPH: le réseau cadencé par les données

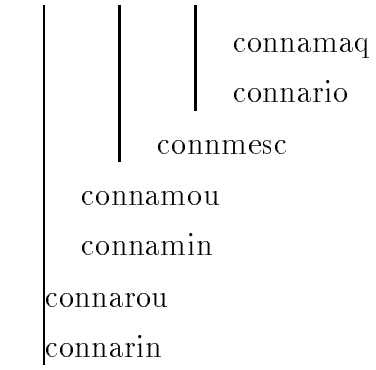
Nous décrirons ici brièvement les modules composant la partie de *A*GRAPH implantée sur la station graphique. La liste ci-dessous traduit leur hiérarchie, les modules `conn...` sont destinés à établir les connections.

B.1 Hiérarchie de différents modules





Connections



B.2 Description succinte

On retrouve ici l'organigramme général de la figure 2.4. Remarquons que dans la liste ci-dessus, les connections figurent après les modules, ceci ayant été fait dans l'esprit d'une programmation modulaire décrite au chapitre 3.

- `fk` et `peri` gèrent les boutons, touches et périphériques.
- le réalisateur (`real`) aiguille ces informations via `aigreal` vers le metteur en scène (`mesc`), le cameraman (`came`) ou l'animateur (`anim`).
- la mise en scène est aussi une question d'aiguillage (`aigmesc`), l'impresario (`ario`) dirige les acteurs et éventuellement les maquille (`maqu`) en les coloriant.
- le cameraman (`came`), forcément aiguilleur (`aigcame`), définit la fenêtre d'affichage (`port`), le type de vue `solo`, vue gauche, vue droite ou stéréo, il (`wifo`) choisit

le mode de projection parallèle (`wind`) ou perspective (`fovi`), il règle les plans avant et arrière (`avar`), déplace la caméra (`from`) et l'oriente (`look`), il procède aux opération de clipping (`clip`), de rotation du plateau tournant (`rota`) et de travelling (`tran`). La position et l'orientation de la caméra peut être visualisée sur l'écran par `refe`. Les connections sont faites par `conncame`, la camera filme tout objet graphique cité dans `applcame`.

- l'animateur (`anim`) active les acteurs; quelquefois par l'intermédiaire d'une boîte de vitesse (`boit`).
- la datation des scènes est faite par le module `scri`, l'affichage du numéro par `afnr`.
- le film commence ... `clap`!

Annexe C

AGRAPH: Programmation de la caméra

C.1 Introduction

Nous présentons ici les modules servant à diriger la caméra virtuelle qui permet de définir la manière dont les objets graphiques de la scène doivent affichés à l'écran.

C.2 Fonctionnnalité de la caméra

Par *caméra* nous entendons toutes les opérations conduisant à l'affichage d'une image à savoir:

- définition de la fenêtre d'affichage
- choix de la vue, mono, stéréo, vue gauche ou droite
- type de projection
- règlage des paramètres de prises de vue
- orientation et position de la caméra par rapport à la scène

A titre d'exemple nous décrirons précisement les modules `port`, `fovi` et `clip`.

C.3 Organisation générale

C.3.1 Imbrication

Voici le diagramme d'imbrication des différents modules composant la caméra.

L'objet `port` est affiché, il fait référence aux objets `wifo_g`, `wifo_d`, `solo_g`, `solo_m` et `solo_d`. `wifo_g` fait lui même référence à `fovi_g` et `fovi_` etc.; l'objet `champ` contient les acteurs et décors suceptibles d'entrer dans le *champ* de vision de la caméra.

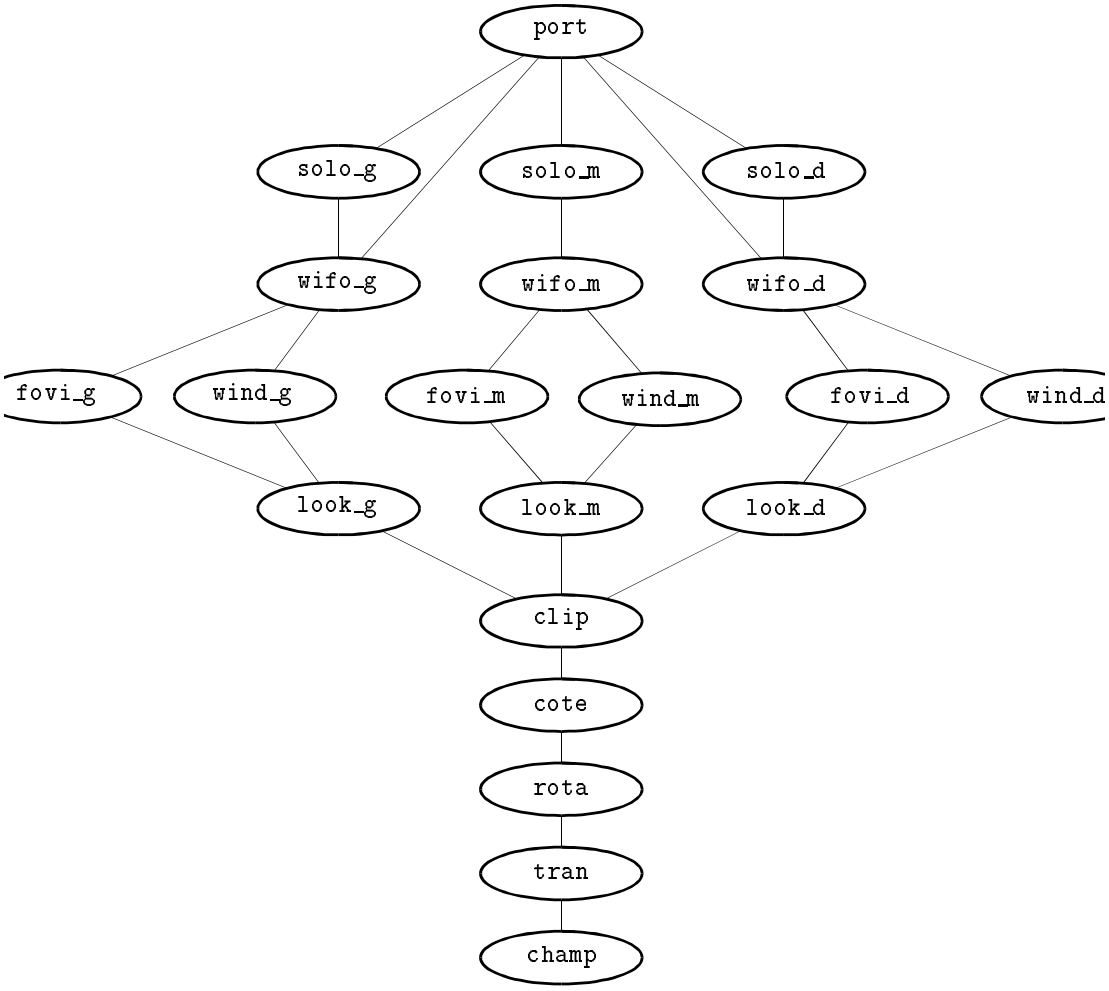


Figure C.1: Imbrications des modules de la caméra.

C.4 Etude précise de quelques modules

C.4.1 La fenêtre d'affichage: PORT

```
port:=begin_structure
        lod:=set level_of_detail to 0 ;
        pm:=if level_of_detail # 1 then port_m ;
        ps:=if level_of_detail = 1 then port_s ;
    end_structure;

port_m      :=viewport horizontal=-1:1
                       vertical=-1:1 applied to port_m_appl ;
port_m_appl:=instance of nil;

port_s      :=instance of port_g,port_d ;

port_g      :=viewport horizontal=-1:0
                       vertical=-0.5:0.5 applied to port_g_appl ;
port_d      :=viewport horizontal= 0:1
                       vertical=-0.5:0.5 applied to port_d_appl ;

port_g_appl :=instance of nil;
port_d_appl :=instance of nil;
```

Remarquons que les ordres `applied to` s'appliquent à des objets graphiques vides (`nil`), qui seront mis à jour par les commandes citées ci-dessus en C.3.1. Cette manière de procéder n'a été adoptée que par un souci de cohérence entre tous les programmes que nous avons écrits; souvent ces ordres d'instanciation ne sont pas connus lors de l'écriture des programmes et peuvent même éventuellement être définis par programme.

Le port d'affichage est différent suivant que l'on affiche une vue mono (ou gauche ou droite pour les photos stéréo), ou une vue stéréo, dans ce cas la taille

des fenêtres d'affichage gauche et droite est variable en fonction de l'ecartement des deux images souhaité par l'utilisateur. Ceci est réalisé par l'intermédiaire du réseau de fonctions cadencé par les données présenté figure C.2.

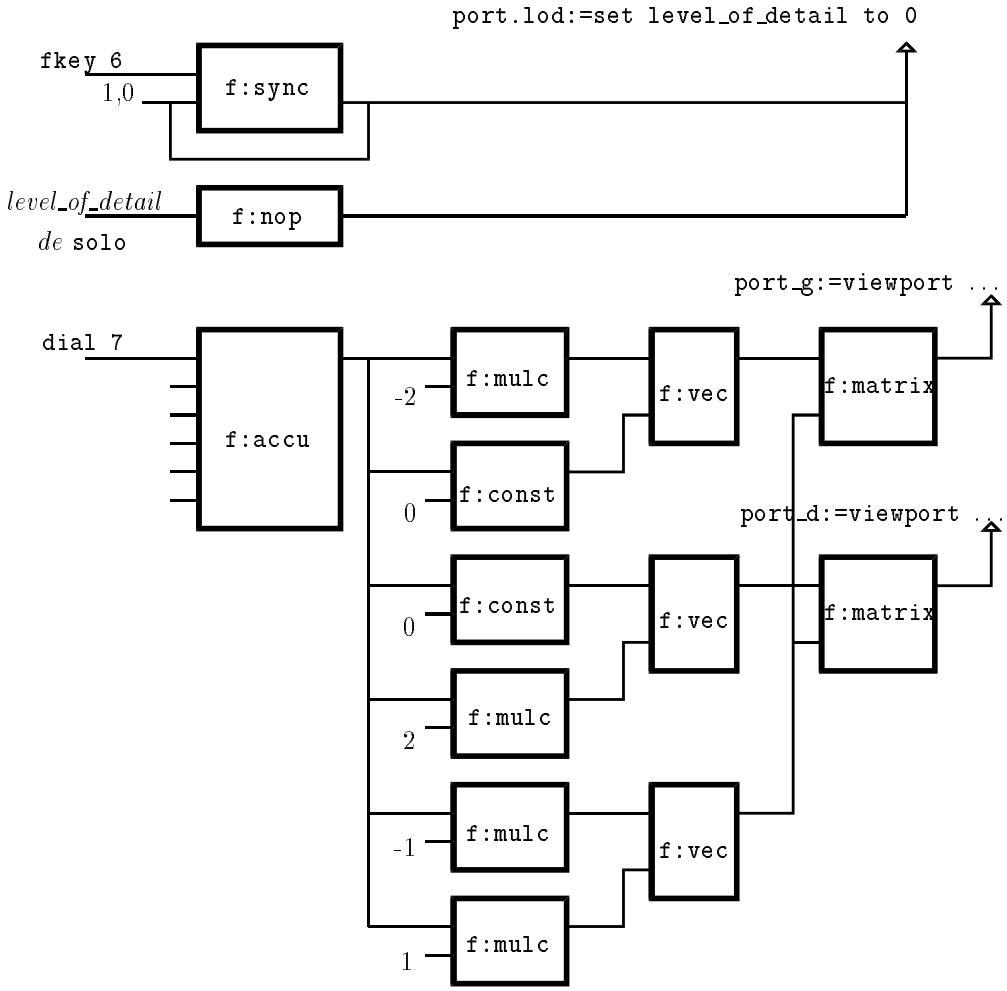


Figure C.2: Le réseau `port` permet de choisir le type de vue, mono, gauche, droite ou stéréo (touche `f5`), ainsi que l'écartement des deux images de la vue stéréoscopique (boutons 7 du caméraman).

C.4.2 La projection perspective: FOVI

```
fovi_m:=field_of_view 50.
                front boundary =0
                back boundary =2 applied to fovi_m_appl ;

fovi_g:=field_of_view 50.
                front boundary =0
                back boundary =2 applied to fovi_g_appl ;

fovi_d:=field_of_view 50.
                front boundary =0
                back boundary =2 applied to fovi_d_appl ;

fovi_m_appl:=instance of nil;
fovi_g_appl:=instance of nil;
fovi_d_appl:=instance of nil;
```

Les trois branches sont nécessaires pour distinguées les vues mono, gauche et droiute. Tous les paramètres définissant la commande `field_of_view` peuvent être modifiés par le réseau représenté figure C.3.

C.4.3 Le module CLIP

Il est possible de ne pas visualiser les parties d'objets situées en deçà et au delà des plans avant et arrière définis par la distance de mise au point et la profondeur de champ. Par analogie, c'est comme s'il était possible de ne pas avoir sur l'image les parties floues d'un objet photographié.

```
clip:=set depth_clipping ON appl to clip_appl ;

clip_appl:=instance of nil;
```

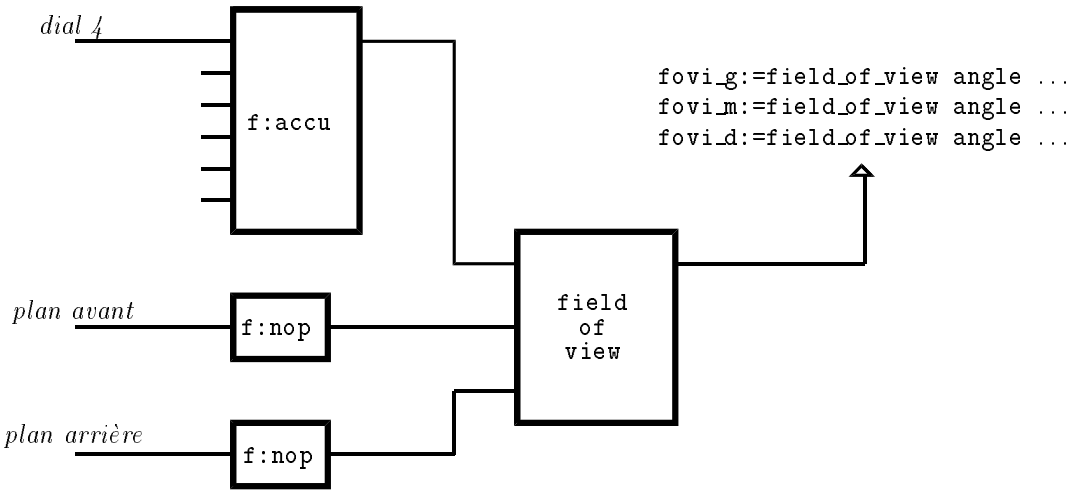


Figure C.3: Le réseau `fovi` permet le règlage de la focale de l'objectif (angle de *zoom*) et effectue la mise à jour des positions des plans avant et arrière prises en compte par l'intermédiaire du réseau `avar`.

Le réseau de la figure C.4 permet de modifier la valeur de *depth_clipping* de ***ON*** à ***OFF*** et inversement.

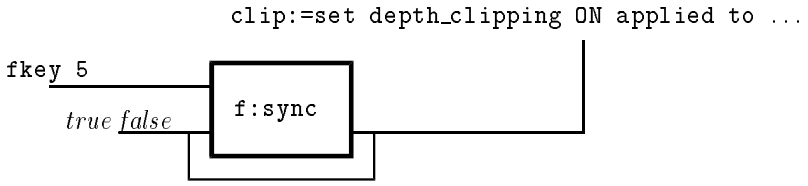


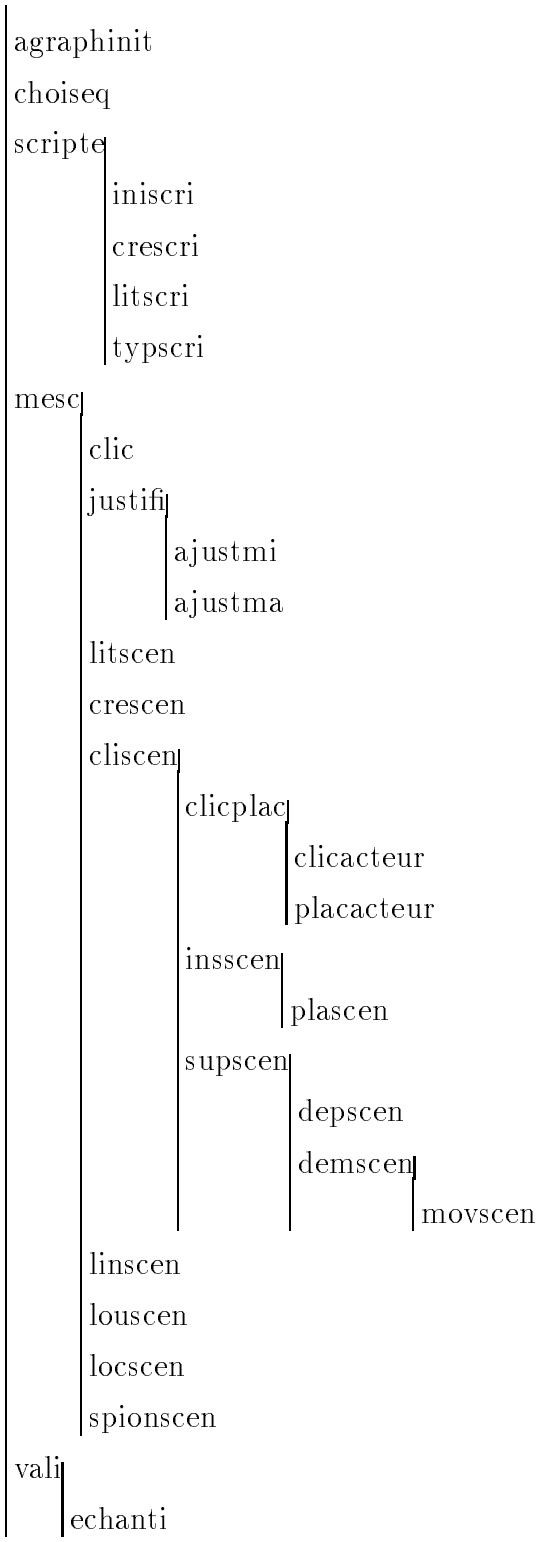
Figure C.4: Le réseau `clip` permet d'activer ou de désactiver l'affichage de parties d'objets situées en deçà et au delà des plans de coupe avant et arrière. Par la touche `f 6` on envoie ***OFF*** (*false*), puis ***ON*** (*true*), puis ***OFF*** (*false*) etc..

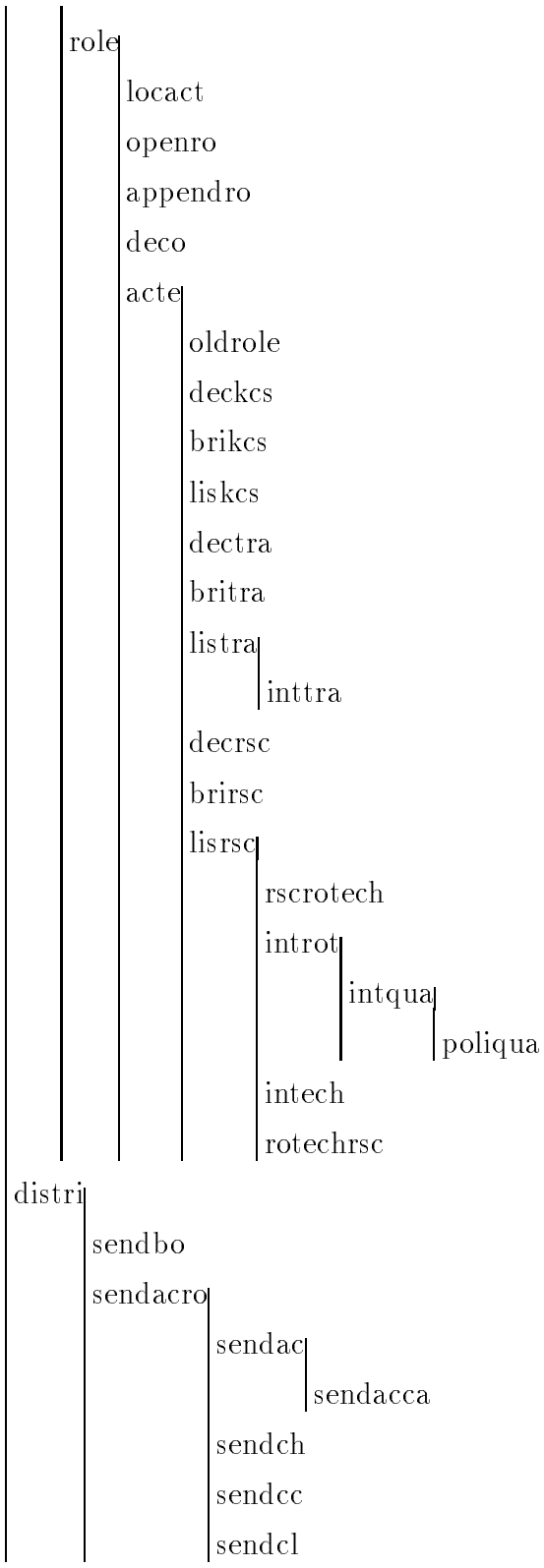
Annexe D

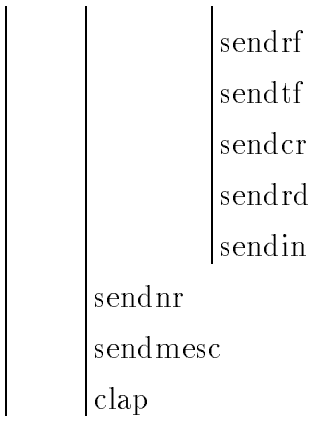
AGRAPH: modules implantés sur l'ordinateur hôte

Nous décrirons ici brièvement les modules FORTRAN composant la partie de *A*GRAPH implantée sur l'ordinateur hôte.

D.1 Hiérarchie de différents modules




role
locact
openro
appendro
deco
acte
oldrole
deckcs
brikcs
liskcs
dectra
britra
listra
inttra
decrsc
brirsc
lisrsc
rscrotech
introt
intqua
poliqua
intech
rotechrsc
distri
sendbo
sendacro
sendac
sendacca
sendch
sendcc
sendcl



D.2 Description succinte

- Après initialisation générale par `agraphinit`, l'utilisateur choisit la séquence à réaliser grâce à `choiseq`.

- La détermination du nombre d'acteurs, leur manière de jouer, sont pris en compte par les modules de création d'un nouveau script (`crescri` et `iniscri`), chargement ou modification d'un ancien script (`litscri`), interrogation et affichage (`typscri`).

- La partie mise en scène (`mesc`) consiste à receptionner les informations relatives aux scènes-clés issues de la station graphique (modules `clic`), en modifiant éventuellement leur date par `justifi, ajustmini` et `ajustmaxi`, celles-ci sont stockées (`clicscen`) pour être éventuellement réutilisées plus tard par `litscen`; ces scènes-clés sont insérées par `insscen` dans la liste ordonnée de scènes existantes, elles peuvent également en être supprimées par `supscen`.

  Une position particulière d'un acteur peut être mémorisée par `clicacteur` pour une éventuelle réutilisation ultérieure par `placacteur` lors de la réalisation d'une autre séquence.

  Toutes ces opérations sont réalisées à l'aide des modules de manipulation de scènes `spionscen`, `linscen`, `locscen` et `louscen`.

- L'interpolation de ces scènes-clés est faite par le module intervaliste `vali`. Celui-ci, après calcul des paramètres d'échantillonage par `echanti` en fonction du script et des dates des scènes-clés, calcul les rôles des acteurs. Trois sortes de rôles sont créés suivant que l'acteur est un **déc**or, ou joue un rôle qui doit être **lis**se ou **bri**sé (constant entre deux scènes-clés). Les paramètres de contraste **k**, de couleur **c**, de saturation **s**, ainsi que la translation **tra**, l'orientation **rot**, et le facteur d'échelle **ech**, sont traités par les modules `deckcs`, `brikcs`, `liskcs`, et `dectra`, `britra`, `listra` etc.. Remarquons que la rotation **rot** et le facteur d'échelle **ech** peuvent être combinés en une matrice **rsc**.

- les rôles peuvent être renvoyés à la station graphique par `distri`. Il y a réinitialisation des réseaux de rôles par l'intermédiaire des modules de chargement `sendbo` pour la boîte de vitesse, `sendac` pour les acteurs, `sendch` pour les réseaux *chef d'orchestre*, rôle, clic et collecte de mise en scène.

- silence on tourne ... `clap`!

Liste des figures

Bibliographie

[Ackerman 1982] Ackerman, W.B., "Data Flow Langages", *IEEE Computer*, Volume **15**, Number 2, February 1982

[Ackerman et Dennis, 1979] Ackerman, W.B. and Dennis, J.B., "VAL-A Value Oriented Algorithmic Language: Preliminary Reference Manual", *Technical Report* TR-218, Laboratory for Computer Science, MIT, Cambridge, Massachussets, June 1979

[Agerwala et Arwind, 1982] Agerwala, T. and Arwind, "Data Flow Systems" (Guest Editor's Introduction) *IEEE Computer*, Volume **15**, Number 2, February 1982

[Allan 1980] Allan, S.J. and Oldehoeft, A.E., "A Flow Analysis Procedure for the Translation of High-Level Languages to a Data Flow Language", IEEE Transaction on Computers, Volume **c-29**, Number 9, September 1980.

[Altmann 1986] "Rotations, Quaternions and Double Groups", Altmann, S.,L., Clarendon Press, Oxford, 1986

[Amerein 1988] Amerein, B., "Modélisation et Représentaion Dynamique de Macromolécules Biologiques", Thèse de l'Université Louis Pasteur de Strasbourg, Mars 1988

[Amerein et Spehner, 1989] Amerein, B. et Spehner, J.C., "Un algorithme de déformation interactive utilisant le cadencement par les données", *Revue Internationale de CFAO et d'Inforgraphie*, Volume **4**, Numéro 4, pp 59-79, 1989

[Amerein et Ripp, 1986] Amerein, B. and Ripp, R., "AGRAPH: a Program to build up Animation Films", *5th International Meeting of the Molecular Graphics Society*, Cap d'Agde, France, 11-14 avril 1986

[Amerein *et al.*, 1987] Amerein, B., Ripp, R. and Dumas, P., "PUCK: A Real Time Modification of Sugar Pucker on a PS300", Journal of Molecular Graphics, Volume **5**, Number 4, pp 184-189, 1987

[Andrews et Schneider, 1983] Andrews, G.R. and Schneider, F.B., "Concepts and Notations for Concurrent Programming", *ACM Computing Surveys*, Volume **15**, Number 1, March 1983

[Apéry 1986] Apéry, F., "La surface de Boy", *Advances in Mathematics*, Volume **61**, Number 3, September 1986

[Apéry 1987] Apéry, F., *Models of the Real Projective Plane*, Vieweg 1987

[Arwind et Gostelow, 1982] Arwind and Gostelow, K.P., "The U-Interpreter", *IEEE Computer*, Volume **15**, Number 2, February 1982

[Arwind *et al.*, 1978] Arwind, Gostelow, K.P. and Plouffe, W., "An Asynchronous Programming Language and Computing Machine", *Technical Report* TR-114a, Department of Information and Computer Science, University of California, Irvine, December 1978

[Backus 1978] Backus, J., "Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs.", *Communication of the ACM* Volume **21**, Number 8, August 1978

[Badler 1987] Badler, N.I., "Articulated Figure Animation", Guest Editor's Introduction, *IEEE Computer Graphics and Applications*, Volume **7**, Number 6, pp 10-11, June 1987

[Badler *et al.*, 1987] Badler, N.I., Manoochehri,K.H. and Walters, G., "Articulated Figure Positionning by Muliple Constraints", *IEEE Computer Graphics and Applications*, Volume **7**, Number 6, pp 28-38, June 1987

[Barsky 1984] Barsky, B.A., "Exponential and Polynomial Methods for Applying Tension to an Interpolating Spline Curve", Computer Vision, Graphics and Image Processing, Volume **27**, Number 1, pp 1-18, July 1984

[Bartels *et al.*, 1987] Bartels, R.H., Beatty, J.C. and Barsky, B.A., "An Introduction to Splines for Use in Computer Graphics & Geometric Modeling", Morgan Kaufmann Publishers, Inc.

[Bass] Bass, J., "Nombres complexes et quaternions", Cours de Mathématique, Tome 1, Fascicule 1, pp 74-80, Masson

[Berger *et al.*, 1987] Berger, P., Comte, D., Hifdi, N., Pelois, B. et Syre, J.C., "Le système LAU: un multiprocesseur à assignation unique", *Techniques et Sciences Informatiques*, Volume **1**, Numéro 1, 1982

[Berry *et al.*, 1987] Berry, G., Couronné, P. et Gonthier, G., "Programmation synchrone des systèmes réactifs: le langage ESTEREL", *Techniques et Sciences Informatiques*, Volume **6**, Numéro 4, 1987

[Bertin 1977] Bertin, J., "La Graphique et le traitement graphique de l'Information", *Nouvelle bibliothèque scientifique*, Flammarion

[Bezier 1987] "Courbes et surfaces", Collection *Mathématiques et CAO*, Tome 4, Hermes, 1987

[Blundel et Johnson, 1976] Blundel, T.L. and Johnson, L.N., "Protein Crystallography", *Academic Press*, p 133, 1976

[Booch 1986] Booch, G., "Object-Oriented Development", *IEEE Transactions on Software Engeneering* Volume **SE-12**, Number 2, pp 211-221, February 1986

[Brodman et Netravali, 1988] Broadman, L.S. and Netravali, A.N., "Motion Interpolation by Optimal Control", Computer Graphics, Volume **22**, Number 4, pp 309-315, August 1988

[Brotman 1988] Brotman, L.S., "Motion Interpolation by Optimal Control", *Computer Graphics*, Volume **22**, Number 4, August 1988

[Calvert 1988] Calvert, T., "The Challenge of Human Figure Animation", *Graphics Interface '88 Proceedings*, Edmonton, Alberta, June 1988

[Chadwick et Parent, 1988] Chadwick, J. and Parent, R., "Critter Construction: Developing Character for Computer Animation", *PIXIM Conference Proceedings*, Volume **21**, Number 1, 1988

[Chevrier et Ripp, 1986] Chevrier, B. and Ripp, R., "Program for the Visualization and Interactive Study of Molecules on a Calligraphic Display System", Journal of Molecular Graphics, Volume **4** Number 4, December 1986

[Church 1941] Church, A., "The Calculi of Lambda Conversion", *Ann. Math. Studies*,Volume **6**, Princenton University Press, Princeton, N.J., 1941

[Colonna 1987] Colonna, J.F., "Le cinéma assisté par ordinateur", *La Recherche*, Numéro 191, pp 1038-1048, septembre 1987

[Computer Architecture 1985] "The 12th Annual International Symposium on Computer Architecture", *SIGARCH Newsletter* Volume **13**, Issue 3, june 1985

[Davis 1978] Davis, A.L., "Data Driven Nets: A Maximally Concurrent Procedural, Parallel Process Representation for Distributed Control Systems", *Technical Report* UUCS-78-108, Computer Science Department, University of Utah, Salt Lake City, 1978

[Davis et Keller, 1982] Davis, A.L. and Keller, R.M., "Data Flow Program Graphs", *IEEE Computer*, Volume **15**, Number 2, February 1982

[Davis] Davis, A.L., "An Introduction to Data-Driven Programming Methodology for PS300 Users", Evans & Sutherland Computer Corporation, P.O. Box 8700, Salt Lake City, Utah 84108

[de Casteljau 1987] de Casteljau, P., "Les quaternions", Edition Hermes

[de Reffye *et al.*, 1988] de Reffye, P., Edelin, C., Françon, F., Jaeger, M. and Puech, C., "Plant Models Faithfull to Botanical Structure and Developpement", Computer Graphics, Volume **22**, Number 4, pp 151-158, August 1988

[DeRose *et al.*, 1989] DeRose, T.D., Bailey, M.L., Barnard,B., Cypher,R., Dobrikin, D., Ebeling, C., Konstandinidou, S., McMurchie, L., Mizrahi, H., and Yost, B. "Apex:two architecture for generating parametric curves and surfaces", The Visual Computer, Volume **5**, Number 5, pp 264-276.

[Delesalle *et al.*, 1988] Delesalle, L., Colonna, J.F. et Fantin, M., "Les premies pas de l'image intelligente", *La Recherche*, Numéro 196, février 1988

[Dennis 1980] Dennis, J.B., "Data Flow Supercomputer", *IEEE Computer*, Volume **13**, Number 11, pp 48-56, November 1980

[Dufaut et Ripp, 1980] Dufaut, M. and Ripp, R., "Real Time Detection of Defective Objects in a Production Line By Vision Sensor", Congrès *EUSIPCO 80*, Lausanne, Suisse, septembre 1980

[Dufourd 1988] Dufourd, J.-F., "Vers un cadre unique pour spécifier et construire des programmes" *Techniques et Sciences Informatiques*, Volume **7**, Numéro 3, 1988

[Dumas et Ripp, 1986] Dumas, P. and Ripp, R., "A Real Time Interactive Graphics Program to Determine Crystal Orientation for the Analysis of Oscillation Diffractions Photographs"' Journal of Applied Crystallography, Volume **19**, Part 1, February 1986

[Duncan 1990] Duncan, R., "A Survey of Parallel Computer Architectures", *IEEE Computer Graphics and Applications*, Volume **23**, Number 2, pp 5-16, February 1990

[Dupuis et Guilbert, 1990] Dupuis, C. et Guilbert, P., "Vue d'ensemble du système graphique CIBOG", *Techniques et Sciences Informatiques*, Volume **9**, Numéro 3, 1990

[Euler 1758] Euler, L., "Du mouvement de rotation d'un corps solide autour d'un axe variable", *Opera Omnia, Ser. secunda*, Volume 8, Orell Füsli Turici, Lausannae, 1758

[Foley 1987] Foley, J.C., "Les communications entre l'Homme et l'ordinateur", *Pour la Science*, Numéro 122, pp 74-82, décembre 1987

[Foley et van Dam, 1982] Foley, J.C. and van Dam, A., "Fundamentals of Interactive Computer Graphics", Addisson Wesley Publishing Company

[Forsythe *et al.*, 1977] Forsythe, G.,E., Malcolm, M.,A. and Moler, C.,B., "Computer Methods for Mathematical Computations", Prentice-Hall, Englewood Cliffs, New Jersey, 1977

[Gajski *et al.*, 1982] Gajski, D.D., Padua, D.A. and Kuck, D.J., "A Second Opinion on Data Flow Machines and Langages", *IEEE Computer*, Volume **15**, Number 2, February 1982

[Gançarski 1988]
Gançarski,P., "Animation par Acteurs-Script", Thèse de l'Université Louis Pasteur de Strasbourg, décembre 1988

[Gaudiot 1985] Gaudiot, J.L., "Methods for Handling Data Structures", *IEEE SIGARCH Newsletter*, Volume **13**, Issue 3, The 12^th Annual Internationnal Symposium on Computer Architecture, June 1985

[Gelernter 1986] Gelernter, D., "Domesticating Parallelism", *IEEE Computer* 1986

[Getto et Breen, 1990] Getto, P. and Breen, D., "The Clockworks: An Object-Oriented Architecture for a Computer Animation System", *The Visual Computer*, Number 6, pp 79-92, 1990

[Graefe 1883] Graefe, F., "Vorlesung uber die Theory der Quaternionen", B.G. Teubner Verlag

[Haeberli 1988] Haeberli, P.E., "ConMan: A Visual Programming Language for Interactive Graphics", Computer Graphics, Volume **22**, Number 4, pp 103-111, August 1988

[Hamilton 1866] Hamilton, W.,R., "Elements of Quaternions", Chelsea Publishing Company, Newyork, N. Y., 1969

[Hanrahan et Sturman, 1985] Hanrahan, P. and Sturman, D., "Interactive Animation of Parametric Models", *The Visual Computer*, pp 260-266, 1985

[Hedelman 1984] Hedelman, H., "A Data Flow Approach to Procedural Modeling", *IEEE Transaction on Computer and Application*, pp 16-26, January 1984

[Hewitt *et al.*, 1973] Hewitt, C., Bishop, B. and Steiger, R., "A Universal Modular Actor Formalism for Artificial Intelligence", *Proceedings of the Internationnal Joint Conference on Artificial Intelligence*, pp 235-245, 1973

[Hoare 1978] Hoare, C.,A.,R., "Communicating Sequential Processes", *Communication of the ACM*, Volume **21**, Number 8, Auguste 1978

[Hofstadter 1979] Hofstadter, D., "Goedel, Escher, Bach: Les Brins d'une Guirlande Eternelle", *Interédition*

[Hologramme 1987] Ripp, R. en collaboration avec Mellet, M. et la société *Hologrammes Industrie*, Hologramme de synthèse de la Cardiotoxine de venin de serpent en couverture du mensuel *Biofutur*, Numéro 62, décembre 1987

[Hudak 1989] "Conception, Evolution and Application of Functional Programming Languages", *ACM Computing Surveys*, Volume **21**, Number 3, September 1989

[Jones 1982] Jones, T.A., "*Frodo*: a graphics fitting program for macromolecules", *Computationnal Crystalography*, Sayre, D., Editors, Clarendon Press, Oxford, pp 303-317, 1982

[Kochanek et Bartels, 1984] Kochanel, D.,H.,U. and Bartels, R.,H., "Interpolating Splines with Local tension, Continuity and Bias Control", *Computer Graphics-SIGGRAPH'84 Conference Proceedings*, Volume 18, Number 3, pp 33-41, July 1984

[Knuth 1986] Knuth, D., "TEX: Computer & Typesetting", Addison-Wesley

[Lafon 1975] Lafon, J.,C., "Sur le produit de deux quaternions", *C. R. Académie des Sciences*, A., FR., DA. Volume 280, Numéro 10, pp 665-668, 1975

[Lamport 1986] Lamport, L., "LaTeX: A Document Preparation System", *User's Guide and Reference Manual*, Addison-Wesley

[Laporte et Magnenat-Thalmann, 1984] Laporte, G. et Magnenat-Thalmann, N., Applications du graphisme par ordinateur. *Gaëtan Morin* éditeur

[Lasseter 1987] Lasseter, J., "Principles of Traditionnal Animation Applied to 3D Computer Animation", *ACM Siggraph '87 Conference Proceedings*, Volume **21** Number 4, 1987

[Le Borgne 1987] Le Borgne, M., "Quaternions et contrôle sur l'espace de quaternions", Rapport de Recherche *INRIA*, Numéro 751, novembre 1987

[Lozover et Preiss, 1983] Lozover,O. and Preiss,K., "Automatic Construction of a Cubic B-Spline Representation for a General Curve", Computers and Graphics, Volume **17**, Number 2, pp 149-153, 1983

[Lucas 1970] Lucas, M., "Production de dessins animés à l'aide d'un terminal graphique", Journées *Systèmes de Traitement Graphique*, IMAG, Grenoble, Avril 1970

[Luciani et Cadoz, 1984] Luciani, A. and Cadoz, C., "Modélisation et animation gestuelle d'objets, le système ANIMA", *Premier colloque Image, CESTA-GRETSI.* Biarritz, Mai 1984

[McCarty 1963] McCarthy, J., "A basis for a mathematical theory of computation", *Computer Programming and Formal Systelms*, North-Holland, The Netherlands, pp 33-70, 1963

[Mackay 1983] Mackay, A.,L., "Quaternion Transformation of Molecular Orientation", *Acta Crystallographica*, A**40**, pp 165-166, 1983

[Magnenat-Thalmann et Thalmann, 1985/a] Magnenat-Thalmann, N. and Thalmann, D., "Computer Animation *Theory and Practice*", Springer Verlag, 1985

[Magnenat-Thalmann et Thalmann, 1985/b] Magnenat-Thalmann, N. and Thalmann, D., "Subactor Data Type as Hierarchical Procedural Models for Computer Animation", *Eurographics'85*, pp 121-128.

[Magnenat-Thalmann et Thalmann, 1987] Magnenat-Thalmann, N. and Thalmann, D., "The Direction of Synthetic Actors in the film *Rendez-vous à Montréal*", *IEEE Computer Graphics and Application*, Volume **7**, Number 12, 1987

[Magnenat-Thalmann *et al.*, 1988] Magnenat-Thalmann, N., Laperrière, R. and Thalmann, D., "Joint-Dependent Local Deformation for Hand Animation and Object Grasping", *Graphics Interface '88 Proceedings*, Edmonton, Alberta, June 1988

[Matelan 1985] Matelan, N., "The Flex/32 Multicomputer" *SIGARCH Newsletter*, Volume **13**, Issue 3, june 1985 pp 209-213

[Moissinac 1984] Moissinac, J.C., "Aides informatiques à la réalisation de dessins animés", Thèse de docteur-ingénieur en informatique, Ecole Nationale Supérieure des Mines de Saint-Etienne

[Moras *et al.*, 1980] Moras,D., Comarmond, M.B., Fisher, J., Weiss, R., Thierry, J.C. Ebel, J.P. and Giegé, R., "Crystal Structure of Yeast $tRNA^{Asp}$", *Nature*, Volume **288**, 18/25, December 1980

[Moras et Bergdoll, 1988] Moras, D. and Bergdoll, M., "Packing and Molecular Interactions in tRNA Crystals" *Journal of Crystal Growth*, Volume 90, pp 283-294, North-Holland, Amsterdam, 1988

[Morin 1978] Morin, B., "Equations du retournement de la sphère", *CRAS*, série A, t. 287,pp 879-882, Paris, 1978

[Morin et Petit, 1979] Morin, B. et Petit, J.P., "Le retournement de la sphère", *Pour la Science*, numéro 15, pp 34-49, 1979

[Mortenson 1985] Mortenson, M.E., *Geometric Modeling*, John Wiley & Sons, Publishers,Inc.

[Moulinier 1990] Moulinier, L., "Contribution à l'affinement de la structure cristallographique d'un complexe DNA-(Lys Trp Lys)", Rapport de stage du magistère de chimie-biologie, Université Louis Pasteur de Strasbourg, 1990

[Mourey *et al.*, 1990] Mourey, L., Samama, J.P., Delarue, M., Choay, Lormeau, J.C., J., Petitou, M. and Moras, D. "Antithrombin III: structural and functional aspects" *Biochimie*, Volume **72**, pp 599-608, 1990

[Neal 1988] Neal, N., "The Birth of a Hologram", About the cover of *IEEE computer Graphics and Applications*, Volume **8**, Number 4, July 1988

[Numerical Recipes] Press, W.H., Flannery, B.P., Teukolsky, S.A. and Vetterling, W.T., "Numerical Recipes, The Art of Scientific Computing", Cambridge University Press, 1986

[O'Donnel et Olson, 1981] O'Donnel, T.J. and Olson, A.J., "*Gramps* - A Graphics Language Interpreter for Real-Time, Interactive, Three-Dimensionnal Picture Editing and Animation", Computer Graphics, Volume **15**, Number 3, pp 133-142, August 1981

[Odyssée 1985] "How to build a planet", *IEEE Computer Graphics and Applications*, August 1985

[Perrin 1987] Perrin, G.-R., "Programmation parallèle: point de vue sur les langages et les méthodes", *Techniques et Sciences Informatiques*, Volume **6**, Numéro 2, 1987

[Péroche *et al.*, 1988] Péroche, B., Argence, J., Ghazanfarpour, D. et Michelucci, D., "La synthèse d'images", Editions Hermès, 1988

[Peters 89] Peters, J., "Local Generalized Hermite Interpolaion by Quadric C^2 Curves", ACM *Transaction on Graphics*, Volume **8**, Number 3, pp 235-242, July 1989

[Petri 1975] Petri, C.A., "General Net Theory", MIT Project MAC *Conference on Petri Nets and Related Methods*, pp 26-41, July 1975

[Piegl 1987] Piegl, L., "Interactive Data Interpolation by Rationnal Bezier Curves", *IEEE Computer Graphics and Applications*, Volume **7**, Number 4, pp 45-58, April 1987

[Pletincks 1988] Pletincks, D., "The Use of Quaternions for Animation, Modelling and Rendering", *Computer Graphics International*, Genève, 24-27 Mai 1988

[PS300 *Users Manual*] *PS300 Users Manual*, Evans & Sutherland, Salt Lake City, Utah, 1981

[Rees *et al.*, 1990] Rees, B., Bilwes, A., Samama, J.P. and Moras, D., "Cardiotoxine V_4^{II} from *Naja mossambica mossambica*. The Refined Crystal Structure", *Journal of Molecular Biology*, Volume **214**, pp 281-297, 1990

[Reeves 83] Reeves, W.T., "Particle systems-A Technique for Modeling a Class of Fuzzy Objects", ACM *Transaction on Graphics*, Volume **2**, pp 91-108, 1983

[Reynolds 1982] Reynolds, C.W., "Computer Animation with Script and Actors", *Computer Graphics Proceedings*, SIGGRAPH 82, pp 289-296, 1982

[Ripp 1987] Ripp, R., "Représentation graphique de surfaces" dans l'ouvrage de F. Apéry: *Models of the Real Projective Plane*, Vieweg 1987

[Ripp *et al.*, 1991] Ripp, R., Moulinier, L. et Samama, J.P, "PhiPsi: A Computer Graphics Simulation of Polypepide Fragments", ...à paraître.

[Sciences & Techniques, 1984] "Spécial Images de Synthèse: Un nouveau monde créé par l'ordinateur", *Sciences & Techniques*, Numéro hors série, Mai 1984

[Shoemake 1985] Shoemake, K., "Animating Rotation with Quaternion Curves", ACM *Computer Graphics*, Volume **19**, Number 3, pp 245-254, August 1985

[State of the Art 1985] "Commercial Demonstrates State-of-the-Art Computer Animation", *IEEE Computer Graphics and Applications*, April 1985

[Stern 1983] Stern, G., "BBop: a system for 3D key frame figure animation", *SIGGRAPH '83 tutorial*, pp 240-243, 1983

[Stone et DeRose, 1989] Stone, M.C. and DeRose, T.D., "A Geometric Characterization of Parametric Cubic Curves", ACM *Transaction on Graphics*, Volume **8**,Number 3, pp 147-163, July 1989

[Sutherland 1963] Sutherland, I.E., "SKETCHPAD: A Man-Machine Graphical Communication System", AFIPS 23, pp 329-346, 1963

[Tait 1867] Tait, P.,G., "Traité élémentaire des quaternions. Théorie - Applications géométriques", Gauthiers-Villars, 1882

[Temps réel, 1988] "Le temps réel", Rapport établi par le *Groupe de Réflexion Temps Réel du CNRS, Techniques et Sciences Informatiques*, Volume **1**, Numéro 1, 1982

[Thelliez et Toulotte, 1982] Thelliez, S. et Toulotte, J.M., "Grafcet et logique industrielle programmée", Eyrolles, 1982

[Treleaven et Lima, 1982] Treleaven, P. ans Lima, I., "Japan's Fifth-Generation Comuter Systems", *IEEE Computer*, Volume **15**, Number 18, pp78-88, August 1988

[Treleaven *et al.*, 1982] Treleaven, P., Brownbridge, D. and Hopkins, R., "Data-Driven and Demand-Driven Computer Architecture", ACM *Computing Surveys*, Volume **14**, Number 1, pp 93-143, March 1982

[Tyler 1984] Tyler, W.M., "3D Images for the Film Industry", *IEEE Computer Graphics World* July 1984

[Verley 1988] Verley, J.L., "Anneau et algèbres", *Encyclopædia Universalis* Volume **2**, p 184, 1988

[Wadge et Ashcroft] Wadge, W. and Ashcroft, E., "Lucid, the Dataflow Programming Language", Academic Press

[Ware et Jessome, 1988] Ware, C. and Jessome, D.R., "Using the Bat: A Six Dimensionnal Mouse for Object Placement", *Graphics Interface '88 Proceedings*, Edmonton, Alberta, June 1988

[Watson et Gurd, 1982] Watson, I. and Gurd, J., "A practical Data flow Computer", *IEEE Computer*, Volume **15**, Number 2, February 1982

[Westhof *et al.*, 1985] Westhof, E., Dumas, P. and Moras, D., "Crystallographic Refinement of Yeast Aspartic Transfer RNA", *Journal of Molecular Biology*, Volume **184**, pp 119-145, 1985

[Yahia et Gagalowicz, 1989] Yahia, H. and Gagalowicz, A., "Interactive Animation of Object Orientations", PIXIM 89 *Proceedings of the* 2^{nd} *Annual Conference on Computer Graphics*, Paris, pp 265-275, September $25^{th} - 29^{th}$ 1989