Le système ANIMA : éditeur d'objets producteurs d'images, implantation d'algorithmes de simulation temps réel

Aimé Razafindrakoto

THESE

Présentée à

L'UNIVERSITE SCIENTIFIQUE, TECHNOLOGIQUE ET MEDICALE DE GRENOBLE
et à
L'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

pour obtenir le titre de
docteur de l'Université Scientifique, Technologique et Médicale de Grenoble

"Informatique"

par

Aimé RAZAFINDRAKOTO

☆☆☆☆☆

LE SYSTEME ANIMA :

. éditeur d'objets producteurs d'images
. implantation d'algorithmes de simulation temps réel.

OOOOO

Thèse soutenue le 8 janvier 1986 devant la commission d'examen.

L. BOLLIET	**Président**
A. LUCIANI	
A. LUX	**Examinateurs**
J.C. MARTY	

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Président : Daniel BLOCH
Vice-Présidents : B. BAUDELET
H. CHERADAME
R. CARRE
J.M. PIERRARD

Année universitaire 1984-1985

Professeurs des Universités

E.N.S.E.E.G.

BESSON	Jean	LOUCHET	François
BONNETAIN	Lucien	PARIAUD	Jean-Charles
BONNIER	Etienne	RAMEAU	Jean-Jacques
DURAND	Françis	SOHM	Jean-Claude
GUYOT	Pierre	SOUQUET	Jean-Louis

E.N.S.E.R.G.

BARIBAUD	Michel	GENTY	Pierre
BLIMAN	Samuel	GUERIN	Bernard
BUYLE BODIN	Maurice	POUPOT	Christian
CHENEVIER	Pierre	SERMET	Pierre
COHEN	Joseph	ZADWORNY	François
COUMES	André		

E.N.S.I.E.G.

BARRAUD	Alain	JOUBERT	Jean-Claude
BAUDELET	Bernard	JOURDAIN	Geneviève
BLOCH	Daniel	LACOUME	Jean-Louis
BRISSONNEAU	Pierre	LONGEQUEUE	Jean-Pierre
CAVAIGNAC	Jean-François	MASSELOT	Christian
CHARTIER	Germain	MORET	Roger
CHERUY	Arlette	PAUTHENET	René
DURAND	Jean-Louis	PERRET	René
FELICI	Noël	PERRET	Robert
FOULARD	Claude	POLOUJADOFF	Michel
GAUBERT	Claude	SABONNADIERE	Jean-Claude
IVANES	Marcel	SCHLENKER	Claire
JALINIER	Jean-Michel	SCHLENKER	Michel
JAUSSAUD	Pierre		

E.N.S.H.G.

BOIS	Philippe	LESPINARD	Georges
BOUVARD	Maurice	MOREAU	René
LESIEUR	Marcel	PIAU	Jean-Michel

E.N.S.I.M.A.G.

ANCEAU	François	MOSSIERE	Jacques
FONLUPT	Jean	ROBERT	François
LATOMBE	Jean-Claude	SAUCIER	Gabrielle
MAZARE	Guy	VEILLON	Gérard

U.E.R.M.C.P.P.

CHERADAME	Hervé	RENAUD	Maurice
CHIAVERINA	Jean	ROBERT	André
GANDINI	Alessandro	SILVY	Jacques

Professeurs Associés

BLACKWELDER	Ronald	ENSHG
HAYASHI	Hirashi	ENSIEG
PURDY	Gary	ENSEEG

Professeurs à l'Université des Sciences Sociales (Grenoble II)

BOLLIET	Louis
CHATELIN	Françoise

Chercheurs du C.N.R.S.

Directeurs de recherche :

CARRE	René
FRUCHARD	Robert
JORRAND	Philippe
VACHAUD	Georges

Maître de recherche :

ALLIBERT	Michel	JOUD	Jean-Charles
ANSARA	Ibrahim	KAMARINOS	Georges
ARMAND	Michel	KLEITZ	Michel
BINDER	Gilbert	LANDAU	Ioan-Dore
BORNARD	Guy	LASJAUNIAS	Jean-Claude
DAVID	René	MERMET	Jean
DESPORTES	Jacques	MUNIER	Jacques
DRIOLE	Jean	PIAU	Monique
GIGNOUX	Damien	PORTESEIL	Jean-Louis
GIVORD	Dominique	THOLENCE	Jean-Louis
GUELIN	Pierre	VERDILLON	André
HOPFINGER	Emile	SUERY	Michel

Personnalités habilitées à diriger des travaux de recherche
(Décision du conseil scientifique)

E.N.S.E.E.G.

ALLIBERT	Colette	HAMMOU	Abdelkader
BERNARD	Claude	MALMEJAC	Yves (CENG)
BONNET	Roland	MARTIN GARIN	Régina
CAILLET	Marcel	NGUYEN TRUONG	Bernadette
CHATILLON	Catherine	RAVAINE	Denis
CHATILLON	Christian	SAINFORT	(CENG)
COULON	Michel	SARRAZIN	Pierre
DIARD	Jean-Paul	SIMON	Jean-Paul
EUSTATHOPOULOS	Nicolas	TOUZAIN	Philippe
FOSTER	Panayotis	URBAIN	Georges(ODEILLO)
GALERIE	Alain		

E.N.S.E.R.G.

BARIBAUD	Michel	DOLMAZON	Jean-Marc
BOREL	Joseph	HERAULT	Jeanny
CHOVET	Alain	MONLLOR	Christian
CHEHIKIAN	Alain		

E.N.S.I.E.G.

BORNARD	Guy	LEJEUNE	Gérard
DESCHIZEAUX	Pierre	MAZUER	Jean
GLANGEAUD	François	PERARD	Jacques
KOFMAN	Walter	REINISCH	Raymond

E.N.S.H.G.

ALEMANY	Antoine	OBLED	Charles
BOIS	Daniel	ROWE	Alain
DARVE	Félix	VAUCLIN	Michel
MICHEL	Jean-Marie	WACK	Bernard

E.N.S.I.M.A.G.

BERT	Didier	DELLA DORA	Jean
CALMET	Jacques	FONLUPT	Jean
COURTIN	Jacques	SIFAKIS	Joseph
COURTOIS	Bernard		

U.E.R.M.C.P.P.

CHARUEL	Robert

C.E.N.G.

CADET	Jean	NIFENECKER	Hervé
COEURE	Philippe (LETI)	PERROUD	Paul
DELHAYE	Jean-Marc (STT)	PEUZIN	Jean-Claude(LETI)
DUPUY	Michel (LETI)	TAIEB	Maurice
JOUVE	Hubert (LETI)	VINCENDON	Marc
NICOLAU	Yvan (LETI)		

Laboratoires extérieurs

C.N.E.T.

DEMOULIN	Eric	MERCKEL	Gérard
DEVINE	R.A.B.	PAULEAU	Yves
GERBER	Roland		

I.N.S.A. Lyon

GAUBERT	C.

****

ECOLE NATIONALE SUPERIEURE DES MINES DE SAINT-ETIENNE

Directeur	: M. M. MERMET
Directeur des Etudes et de la formation	: M. J. LEVASSEUR
Directeur des Recherches	: M. J. LEVY
Secrétaire Général	: Mle M. CLERGUE

Professeurs de la 1ère Catégorie

COINDE	Alexandre	Gestion
GOUX	Claude	Metallurgie
LEVY	Jacques	Métallurgie
LOWYS	Jean-Pierre	Physique
MATHON	Albert	Gestion
RIEU	Jean	Mécanique-Résistance des matériaux
SOUSTELLE	Michel	Chimie
FORMERY	Philippe	Mathématiques Appliquées

Professeurs de 2ème catégorie

HABIB	Michel	Informatique
PERRIN	Michel	Géologie
VERCHERY	Georges	Matériaux
TOUCHARD	Bernard	Physique industrielle

Directeur de recherche

LESBATS	Pierre	Métallurgie

Maîtres de recherche

BISCONDI	Michel	Métallurgie
DAVOINE	Philippe	Géologie
FOURDEUX	Angeline	Métallurgie
KOBYLANSKI	André	Métallurgie
LALAUZE	René	Chimie
LANCELOT	Francis	Chimie
LE COZE	Jean	Métallurgie
THEVENOT	François	Chimie
TRAN MINH	Canh	Chimie

Personnalités habilitées à diriger des travaux de recherche

DRIVER	Julian	Métallurgie
GUILHOT	Bernard	Chimie
THOMAS	Gérard	Chimie

Professeur à l'UER de Sciences de Saint-Etienne

VERGNAUD	Jean-Maurice	Chimie des Matériaux & chimie industrielle

****

UNIVERSITE SCIENTIFIQUE ET MEDICALE DE GRENOBLE

Année universitaire 1982-1983

Président de l'Université : M. TANCHE

MEMBRES DU CORPS ENSEIGNANT DE L'U.S.M.G.

(RANG A)

SAUF ENSEIGNANTS EN MEDECINE ET PHARMACIE

PROFESSEURS DE 1ère CLASSE

ARNAUD Paul	Chimie organique
ARVIEU Robert	Physique nucléaire I.S.N.
AUBERT Guy	Physique C.N.R.S.
AYANT Yves	Physique approfondie
BARBIER Marie-Jeanne	Electrochimie
BARBIER Jean-Claude	Physique expérimentale C.N.R.S. (labo de magnétisme)
BARJON Robert	Physique nucléaire I.S.N.
BARNOUD Fernand	Biosynthèse de la cellulose-Biologie
BARRA Jean-René	Statistiques - Mathématiques appliquées
BELORISKY Elie	Physique
BENZAKEN Claude (M.)	Mathématiques pures
BERNARD Alain	Mathématiques pures
BERTRANDIAS Françoise	Mathématiques pures
BERTRANDIAS Jean-Paul	Mathématiques pures
BILLET Jean	Géographie
BONNIER Jean-Marie	Chimie générale
BOUCHEZ Robert	Physique nucléaire I.S.N.
BRAVARD Yves	Géographie
CARLIER Georges	Biologie végétale
CAUQUIS Georges	Chimie organique
CHIBON Pierre	Biologie animale
COLIN DE VERDIERE Yves	Mathématiques pures
CRABBE Pierre (détaché)	C.E.R.M.O.
CYROT Michel	Physique du solide
DAUMAS Max	Géographie
DEBELMAS Jacques	Géologie générale
DEGRANGE Charles	Zoologie
DELOBEL Claude (M.)	M.I.A.G. Mathématiques appliquées
DEPORTES Charles	Chimie minérale
DESRE Pierre	Electrochimie
DOLIQUE Jean-Michel	Physique des plasmas
DUCROS Pierre	Cristallographie
FONTAINE Jean-Marc	Mathématiques pures
GAGNAIRE Didier	Chimie physique

.../...

GASTINEL Noël	Analyse numérique - Mathématiques appliquées
GERBER Robert	Mathématiques pures
GERMAIN Jean-Pierre	Mécanique
GIRAUD Pierre	Géologie
IDELMAN Simon	Physiologie animale
JANIN Bernard	Géographie
JOLY Jean-René	Mathématiques pures
JULLIEN Pierre	Mathématiques appliquées
KAHANE André (détaché DAFCO)	Physique
KAHANE Josette	Physique
KOSZUL Jean-Louis	Mathématiques pures
KRAKOWIAK Sacha	Mathématiques appliquées
KUPTA Yvon	Mathématiques pures
LACAZE Albert	Thermodynamique
LAJZEROWICZ Jeannine	Physique
LAJZEROWICZ Joseph	Physique
LAURENT Pierre	Mathématiques appliquées
DE LEIRIS Joël	Biologie
LLIBOUTRY Louis	Géophysique
LOISEAUX Jean-Marie	Sciences nucléaires I.S.N.
LOUP Jean	Géographie
MACHE Régis	Physiologie végétale
MAYNARD Roger	Physique du solide
MICHEL Robert	Minéralogie et pétrographie (géologie)
MOZIERES Philippe	Spectrométrie - Physique
OMONT Alain	Astrophysique
OZENDA Paul	Botanique (biologie végétale)
PAYAN Jean-Jacques (détaché)	Mathématiques pures
PEBAY PEYROULA Jean-Claude	Physique
PERRIAUX Jacques	Géologie
PERRIER Guy	Géophysique
PIERRARD Jean-Marie	Mécanique
RASSAT André	Chimie systématique
RENARD Michel	Thermodynamique
RICHARD Lucien	Biologie végétale
RINAUDO Marguerite	Chimie CERMAV
SENGEL Philippe	Biologie animale
SERGERAERT Francis	Mathématiques pures
SOUTIF Michel	Physique
VAILLANT François	Zoologie
VALENTIN Jacques	Physique nucléaire I.S.N.
VAN CUTSEN Bernard	Mathématiques appliquées
VAUQUOIS Bernard	Mathématiques appliquées
VIALON Pierre	Géologie

PROFESSEURS DE 2ème CLASSE

ADIBA Michel	Mathématiques pures
ARMAND Gilbert	Géographie

.../...

AURIAULT Jean-Louis	Mécanique
BEGUIN Claude (M.)	Chimie organique
BOEHLER Jean-Paul	Mécanique
BOITET Christian	Mathématiques appliquées
BORNAREL Jean	Physique
BRUN Gilbert	Biologie
CASTAING Bernard	Physique
CHARDON Michel	Géographie
COHENADDAD Jean-Pierre	Physique
DENEUVILLE Alain	Physique
DEPASSEL Roger	Mécanique des fluides
DOUCE Roland	Physiologie végétale
DUFRESNOY Alain	Mathématiques pures
GASPARD François	Physique
GAUTRON René	Chimie
GIDON Maurice	Géologie
GIGNOUX Claude (M.)	Sciences nucléaires I.S.N.
GUITTON Jacques	Chimie
HACQUES Gérard	Mathématiques appliquées
HERBIN Jacky	Géographie
HICTER Pierre	Chimie
JOSELEAU Jean-Paul	Biochimie
KERCKOVE Claude (M.)	Géologie
LE BRETON Alain	Mathématiques appliquées
LONGEQUEUE Nicole	Sciences nucléaires I.S.N.
LUCAS Robert	Physiques
LUNA Domingo	Mathématiques pures
MASCLE Georges	Géologie
NEMOZ Alain	Thermodynamique (CNRS - CRTBT)
OUDET Bruno	Mathématiques appliquées
PELMONT Jean	Biochimie
PERRIN Claude (M.)	Sciences nucléaires I.S.N.
PFISTER Jean-Claude (détaché)	Physique du solide
PIBOULE Michel	Géologie
PIERRE Jean-Louis	Chimie organique
RAYNAUD Hervé	Mathématiques appliquées
ROBERT Gilles	Mathématiques pures
ROBERT Jean-Bernard	Chimie physique
ROSSI André	Physiologie végétale
SAKAROVITCH Michel	Mathématiques appliquées
SARROT REYNAUD Jean	Géologie
SAXOD Raymond	Biologie animale
SOUTIF Jeanne	Physique
SCHOOL Pierre-Claude	Mathématiques appliquées
STUTZ Pierre	Mécanique
SUBRA Robert	Chimie
VIDAL Michel	Chimie organique
VIVIAN Robert	Géographie

Je tiens à remercier Monsieur Louis Bolliet pour l'honneur qu'il me fait en présidant ce jury, et pour l'attention bienveillante avec laquelle il m'a suivi. Je tiens à m'excuser d'avoir trop souvent solliciter son aide, qu'il a donnée avec gentillesse, pour faciliter certaines démarches administratives.

Je tiens à remercier particulièrement Annie Luciani et Claude Cadoz de m'avoir accueilli dans leur équipe, d'avoir assuré mon encadrement et de m'avoir fait connaître des expériences enrichissantes sur beaucoup d'aspects de la communication homme-machine.

Je remercie vivement mes collègues qui ont donné leur soutien dans ce travail et en particulier J.L. Florens pour sa compétence à fournir des explications, entre autres dans le domaine de la physique et du traitement de signal, qui me semblaient importants pour la compréhension de certains problèmes.

Je tiens enfin à remercier Monsieur Philippe Jorrand d'avoir accepté dans son laboratoire notre équipe dont le travail sort un peu des sentiers battus, Messieurs Augustin Lux et Jean Charles Marty pour l'honneur qu'il me font en participant à ce jury.

PRESENTATION

INTRODUCTION

PRESENTATION

Ce travail a été effectué a' l' ACROE (Association pour la création et la recherche sur les outils d'expression) sous la responsabilité de Annie Luciani en collaboration avec l'équipe Informatique Musicale et Graphique du Laboratoire d'Informatique fondamentale et d'Intelligence Artificielle (LIFIA / Imag).

Il constitue un aboutissement à nombre de réflexions , tentatives, expérimentations antérieures effectuées au sein de l'équipe, pendant lesquelles ont été introduits les concepts tels que Rapport instrumental, Expérimentation multisensorielle d'objets-instruments, Relation artiste-machine dans un contexte de création d'oeuvres à caractère esthétique utilisant l'ordinateur. Ce travail d'étude et de réalisation illustre en partie le transfert de ces concepts sur des processus informatiques.

Le rapport instrumental évoqué ici consiste en une sorte de dialogue entre un manipulateur et des objets-instruments par lequel le manipulateur tend à identifier les fonctions des objets en établissant expérimentalement des relations entre ses actions et les réponses de ces derniers. Ces relations engagent un investisment sensoriel : les études menées jusqu' ici mettent en jeu la participation de trois canaux perceptifs fondamentaux qui sont le toucher, la vue et l'ouie.

Pour tenter de retrouver cette situation de type instrumental dans un environnement informatique (le musicien face à l'instrument de musique, le marionnettiste face aux marionettes), l'ordinateur est considéré comme un moyen de représentation d'un univers d'objets à caractère multisensoriel et en particulier une source de production d'évènements sensoriels.

La connaissance de notre propre univers physique est sous-jacente à la définition de cet univers d'objet. Cette définition est d'autre part soumise aux besoins de l'artiste et à sa fantaisie quand à la recherche du degré de réalisme ou à la recherche d'exagérations possibles dont il a le contrôle, sur le comportement des objets (ex. définition d'un objet infiniment élastique, variation de la force gravitationnelle appliquée à un objet ...). Des contraintes d'ordre technique sont à prendre en compte pour la détermination de cet univers en raison de la puissance de calcul sollicitée par une simulation mécanique des objets.

Les communications entre l'artiste et les objets seront prises en charge par des dispositifs connectés au calculateur, que manipule

l'artiste et qui tiennent compte de la bilatéralité du canal gestuel (émetteur et recepteur) dans une manipulation gestuelle réelle de ces objets. Ces dispositifs sont appelées des transducteurs gestuels.

La restitution sur écran des comportements dynamiques des objets par l'intermédiaire des transducteurs visuels permet de produire des images animées. Ainsi une exécution cyclique des processus manipulation, simulation, visualisation peut servir de support pour l'activité créatrice dans le domaine visuel.

Ce "retour aux sources" (recherche de la situation instrumentale) n'est pas arbitraire, l'utilisation de l'ordinateur ouvre des horizons nouveaux pour la production d'évènements sensoriels inédits grâce à la variété potentielle des objets pouvant être décrits.

Dans cette approche de la fonction de création par l'ordinateur, les coûts d' experimentation sont quasi-négligeables, une fois que l'artiste concepteur d'images animées a conçu ses objets. Il peut en effet de manière interactive demander leur expérimentation, renouveler facilement et à plusieurs reprises ses experiences, permettant l'exploration systématique des résultats visuels, sans mémorisation intermédiaire. Cette facilité est la contrepartie d'une mise en oeuvre "un peu ardue" d'un modèle de simulation mécanique qui fonctionne en temps reel.

La possibilité de mémorisation des actions du manipulateur ainsi que des objets dans des fichiers permet de restituer des expérimentations antérieures. Ces fichiers peuvent par ailleurs être utilisés pour un fonctionnement en temps différé c'est à dire dans des conditions où le temps nécéssaire à la simulation d'objets relativement complexes empêche un synchronisme réel entre la manipulation et la visualisation.

Le système réalisé actuellement se définit comme une maquette démonstrative et évaluative. Elle prend en charge la description d'objets simples et réunit autour de moyens relativement modestes des conditions essentielles pour une experimentation multisensorielle des objets dans laquelle la boucle geste-vue joue le rôle important. Cela doit servir de point d'ancrage a' l'élaboration d'un systeme plus général mettant en oeuvre des objets complexes, des objets à caractéristiques vibratoires associés à des transducteurs acoustiques et à un essai de généralisation des systèmes de transduction. Des travaux dans le sens de la création d'un tel outil sont en cours au sein de l'équipe. Dans le domaine du son produit par la simulation des phénomenes vibratoires, des références sur les premiers essais concluants existent dans la bibliographie actuelle.

Un des objectifs visés est de permettre l'accessibilité du système à

des non-informaticiens c'est à dire être en mesure de traduire d'une façon transparente un "langage d'artiste" en langage mathématique et informatique. Un aspect non négligeable de ce travail repose en effet sur l'étude et la réalisation d'une interface utilisateur.

Le matériel exploité ici regroupe :
- un mini-ordinateur LSI 11/02 à 32k mots
- un processeur vectoriel AP 120-B
- une console graphique de dialogue + tablette à digitaliser
- des transducteurs gestuels à 1 et 2 degrés de liberté
- un système de visualisation pour des dessins au trait à deux dimensions.

Après une brève synthèse de quelques méthodes utilisées dans le domaine de l'animation assistée par l'ordinateur ,nous évoquerons les fondements théoriques de la démarche et nous essayerons de décrire le cahier de charges pour le système ANIMA . Ce cahier doit prendre en compte l'environnement général permettant d'établir les étapes à suivre par l'expérimentateur depuis la phase de construction des objets jusqu' à leur expérimentation dynamique .

A partir des premières spécifications externes contenues dans ce cahier et d'une "infrastructure" logicielle inexistante, nous tenterons de dégager par étape un certain nombre d' outils adéquats conduisant à une implantation, après un débroussaillage des problèmes algorithmiques. Ce travail tient compte des spécifications récentes définies par Annie Luciani dans son rapport de thèse[12] qui fait une synthèse de l'ensemble des travaux cités au second paragraphe.

INTRODUCTION

1. Aperçu sur la synthèse d'image par ordinateur

Les demandes potentielles en communication graphique (outils de conception, outils de fabrication, outils de marketing, cartographie) ont contribué de façon déterminante à l'avancement des travaux de recherche dans le domaine de la synthèse d'image.

En dehors des besoins d'ordre industriel, des besoins d'ordre socio-culturels qui requièrent des images non moins complexes, apparaissent et accentuent la croissance dans ce domaine. Nous pouvons remarquer à ce niveau la tendance à une utilisation intensive de pages publicitaires, pages vidéotextes, tableaux provenant d'une "peinture sur ordinateur", qui émanent tous de techniques de synthèse d'image.

Cette croissance dans le domaine de l'image est dûe essentiellement à la structuration des processus de visualisation ,et leur implantation sur des dispositifs cablés. L'apparition du processeur graphique comme élément de la structuration est essentielle. Ses fonctions de base (génération de vecteurs, clôture et fenêtrage, génération de caractères, commandes de balayage de l'écran) évoluent peu à peu pour devenir un vrai synthétiseur cablé intégrant une partie des algorithmes dits "classiques" (remplissage de surfaces, calcul de surfaces cachées, perspectives, luminosité...).

La génération des images pleines a par ailleurs profité de la réduction du coût des mémoires servant à la mémorisation des pixels, et de la maîtrise de techniques de visualisation sur tube cathodique multichrome à balayage vidéo, dispositif ayant bénéficié d'une grande distribution au niveau du public.

Devant l'éventail des techniques actuelles il semble important d'établir des cohérences tant au niveau des systèmes produits que des demandes. En effet le foisonnement de ces techniques qui existent en 2D (mémorisation d'images par pictogramme ou vectogramme) et 3D (techniques de lancer de rayon, definition de facettes polygonales, fractales,...) a entrainé l'existance de systèmes spécifiques, rendant difficile la définition de norme, l'usage de standard et les applications pédagogiques. Un essai de systématisation sur le processus de la synthèse d'image (F.Martinez, Imag) constitue une démarche importante dans ce domaine.

2. L'image dynamique face à l'image statique

D'une manière générale l'essor qui a pu être pris dans le domaine de l'image dynamique a été subordonné à celui de l'essor pris dans celui de l'image statique, faisant apparaître l'image dynamique comme un

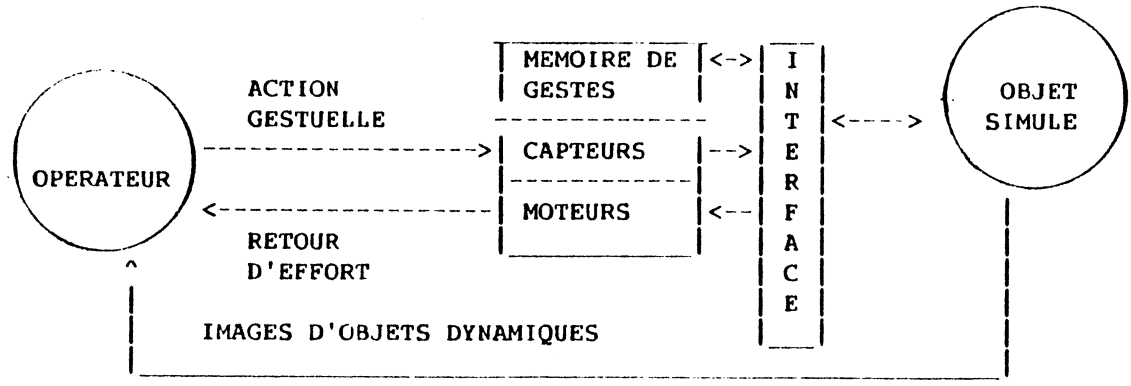
sous-produit de cette dernière, plûtot qu'un domaine à part entière.

L'absence de norme pour la génération d'image statique et l'absence de méthodes systématiques rendent encore plus difficile son utilisation dans un processus dynamique. Dans cette utilisation rentrent en ligne de compte d'une part le calcul de chaque image (saisie de pixels ou saisie de paramètres géométriques) et leur mémorisation, d'autre part l'agencement entre toutes les images.

L'utilisation des dessins au trait est encore nécessaire dans le cas d'affichage rapide en temps réel. L'obtention de ce type d'images de façon continue se heurte pourtant à de nouveaux problèmes, comparativement aux images pleines, en raison de l'utilisation d'un processeur d'entretien nécessitant la gestion des segments d'affichage ("display files") et d'un écran à balayage cavalier généralement de coût élevé. Les systèmes capables de générer rapidement les images pleines existent, comme dans la génération d'images en simulation de vol, mais restent très spécialisés et d'un coût prohibitif [50]. Une difficulté réside en général dans la visualisation rapide des images qui est la synchronisation entre la production des pixels et la conversion de ces derniers en signal analogique.

Dans le domaine d'une animation par ordinateur, la tendance encore à l'heure actuelle à l'utilisation de systèmes incluant des processus analogiques pour la transformation sur l'image dénote l'absence de techniques satisfaisantes issues de l'ordinateur malgré la part de souplesse et de fiabilité attribuée aux systèmes numériques.

Pour nous, nombre de difficultés actuelles semblent inhérentes à la conception même du mouvement dans l'image dynamique. On s'aperçoit que malgré une certaine diversification des résultats, cette conception se fait généralement à un niveau formel, c'est à dire par une composition logico-mathématique d'images d'objets. Cette attitude a pour finalité la recherche des procédés de synthèse rapide intégrant ces transformations (ex. interpolateurs cablés) plutôt que la production d'un mouvement réel par lequel les objets sont sensés être animés. Cette production nécessite alors une simulation par l'informatique des objets et utiliserait en aval de celle-ci les procédés de synthèse d'images comme support à la communication visuelle.



Le propos de notre travail consiste en une validation dans cette dernière approche. Cette situation peut alors amener une remise en question d'un rapport "classique" entre une conception des images animées et l'ordinateur.

Nous évoquons ici, quoique que nous nous en réclamions pas absolument, quatre grandes tendances pour la fabrication d'images en mouvement, ayant servi de références au point de départ de notre recherche. Nous élaborons par la suite quelques critiques sur des points faisant la jonction avec notre préoccupation actuelle tels que la conception du mouvement ou la conception d'une création esthétique émanant de l'utilisation de l'ordinateur.

2.1. Les systèmes hybrides analogiques-numériques.

Ces systèmes ont surtout eu leur succès entre les années 64 et 75 en raison de la vitesse du dispositif analogique pour faire des transformations sur l'image. Des effets comme le grossissement, ou aplatissement... sont obtenus et à des vitesses variables programmables, à partir d'une image source. Un dispositif analogique contrôle dans ce cas la déflexion d' un signal source ,sur les paramètres comme la fréquence, l' amplitude, la luminosité et la couleur. Ce dispositif peut être lui même contrôlé par un calculateur numérique ou par un opérateur, par des accès analogiques (bancs de potentiomètres, "switchs",...). Par exemple, les fonctions type sinus ou delta permettent d'obtenir des formes elliptiques ou polygonales quand elles sont appliquées à un segment de droite. Il est important dans ce cas de pouvoir partitionner l'image source pour faire des déformations locales. Nous citons ici quelques exemples de ces systèmes:

Beflix par Ken Knowlton (1964) [18]
Animac, puis Scanimate par Computer Image Corporation [18]
Caesar [18]

2.2. Les systèmes de production image par image

Ils découlent en général de techniques particulières d'animation cinématographique. Des primitives adaptées d'acquisition de données, de coloration de dessins, permettent la production de l'équivalent des cellulos. Une image peut être plus ou moins complexe suivant les algorithmes de synthèse utilisés (saisie de dessins et gouachage avec mémorisation des pixels ou de segments de droite, images d'objets 3d, ...). Dans la cas des images 3d on assiste surtout à un production de petits films produits par une succession d'images indépendantes qui n'obéissent pas à des règles structurées de succession. Dans le cas des dessins on essaie de définir un langage d'animation permettant de mettre en oeuvre des règles qui s'appliquent sur des entités du

langage. Pour celui-ci l'objectif premier est la réduction de coût de l'animation de techniques traditionnelles car l'ordinateur aide les animateurs à se dégager de tâches fastidieuses et répétitives par sa capacité de faire du calcul automatisé.

Nous énumérons ici quelques règles qui permettent une forme d'automatisation dans la production des dessins, évitant l'exécution manuelle de ces tâches :

- la technique des dessins clés
  Il s'agit là d'édifier le programme qui, à partir de deux dessins extrêmes définis par l'animateur calcule les dessins intermédiaires par une interpolation linéaire ou non linéaire.

- fonctions d'évolution, trajectoires, grilles
  Ces fonctions permettent à partir d'un dessin d'engendrer une séquence par des équations spatio-temporelles : transformations géométriques , suivi par un objet d'une trajectoire définie par l'équation d'une courbe, changements de repère... Le facteur temps permet de donner une illusion d'accélération ou de décélération sur les transformations.

- animation à partir d'une structuration d'objet
  La méthode revient à définir des objets et à les rendre décomposables en objets élémentaires. Une bibliothèque de dessins dans ce cas est associée à chaque objet élémentaire. Un programme d'assemblage permet de reconstituer un objet; un autre programme permet de définir une séquence d'objets assemblés pour produire une animation. On se passe, dans cette optique, de la génération de dessins intermédiaires pour éviter des problèmes d'interpolation. Un choix judicieux de la base de données permet d'obtenir une animation correcte.

Les exemples suivants rentrent dans la catégorie des systèmes de production image par image :

Antics de A. Kitching [10]
Grass, Anima, Anima II, SAS du Computer Graphics Research Group à l'Ohio State University (1975 -> 1982) [18]
Daao [16]
Psyche-anim2 [16]
Safran de F. Martinez (1977) [13]

Quelques remarques générales

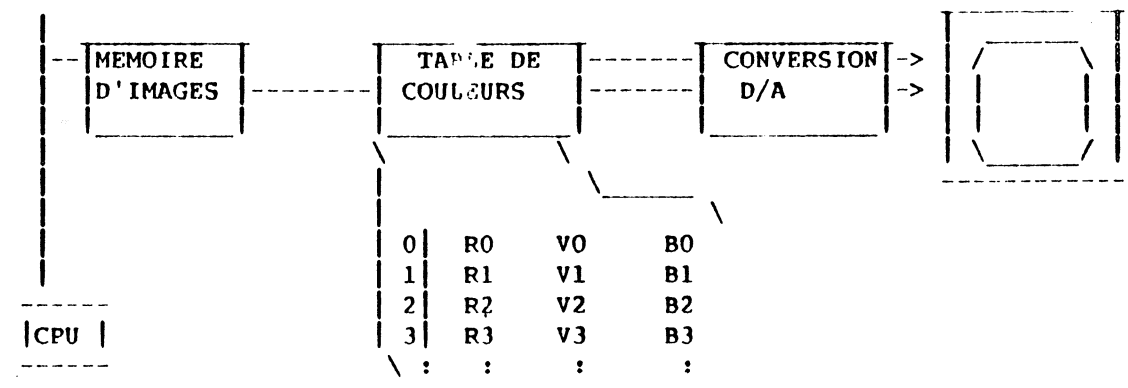
Dans ces différentes techniques, on retrouve des caractéristiques propres à celles de l'animation traditionnelle:

la scène est souvent constituée d'un décor et d'acteurs. Le décor a une définition statique, sur laquelle peuvent être "appliqués" des acteurs qui subissent des modifications, ce qui permet de faire des économies sur le traitement. Une image générée est stockée sur un média intermédiaire telle que film, mémoire de masse ou vidéodisque. Un premier défilement des images est souhaitable pour une appréciation de l'animation. "Retouches, sophistications, intégration dans le décor" se font après cette première phase.

Pour ce qui est du traitement informatique, la projection par l'artiste des scènes en 3d , sur des dessins, pose des problèmes d'insuffisance d'information notamment dans le calcul des parties cachées lors d'un déplacement d'élément. Pour palier cette insuffisance, on rajoute généralement des notions de profondeur, nécessitant l'intervention des traitements en pseudo-3d.

2.3. Animation par changement de couleurs.

Cette technique permet de simuler une animation par un changement de couleur sur image statique. La procédure nécessite l'utilisation d'une table de couleur regroupant les définitions de couleur attribuable à chaque pixel. Un pixel est mémorisé sur n bits dans une mémoire d'image. La table contient alors 2 puissance n valeurs de couleurs. Une fonction délivre une entrée dans la table à partir de la valeur d'un pixel. Des opérations logiques sur la table permettent le passage d'éléments d'une classe de couleur à une autre. La table est à accès rapide, la vitesse de mise à jour est programmée par l'utilisateur pour obtenir la vitesse d'animation désirée.



Des exemples d'application de ce type d'animation sont l'illustration des écoulements d'un fluide ou le déplacement d'objets en météorologie. Comme système on peut citer dans ce domaine le

Superpaint de Xerox [17].

2.4. Images dynamiques

On s'intéresse ici à l'affichage d'images simples permettant une animation continue. En fait, l'affichage dynamique à partir d'une mémoire d'images de 480 * 640 * 8 bits/pixel nécessiterait un flux de 20 Mbits/s pour une vitesse de 10 images/s. Pour réduire ce flux, des contraintes sont établies à plusieurs niveaux :

- réduction de la définition de la couleur,
- réduction de la résolution de l'écran,
- utilisation des dessins au trait au dépens des images pleines. Dans ce cas, le balayage d'une liste de visualisation représentant uniquement des vecteurs réduit considérablement le temps d'affichage,
- réduction du processus de visualisation. Cette réduction peut être introduite en faisant une conversion analogique directe à partir de la mémoire centrale (sans passer par une mémoire d'entretien intermédiaire). Par ailleurs l'utilisation de systèmes à plusieurs buffers d'images réduit généralement les temps de synchronisation entre les images à afficher[2].

L'animation temps réel a surtout été possible jusqu'à aujourd'hui dans des systèmes éducatifs pour une illustration simplifiée de phénomènes dynamiques, régis par équations mathématiques dont les paramètres d'espace et de temps sont connus. On rencontre aussi des applications dans les "jeux vidéo" et la génération d' images plastiques simples représentant des objets 2D mûs par des transformations géométriques .

3. Critique des systèmes d'animation

3.1. Sur l'activité de création

Le consensus établi à l'heure actuelle sur l'utilisation de l'ordinateur est que ce dernier est considéré comme un automate réalisant des tâches fastidieuses, en aval de l'activité créatrice dans l'animation. La fonction de création figure comme expérience empirique intériorisée par l'animateur et projetée sur une succession de dessins ou sur des courbes spatio-temporelles. A cet effet, quelques professionnels de l'animation cinématographique utilisent l'informatique (ceux ayant accepté le coût encore élévé, et la rigueur informatique pour une animation de qualité) ou par des informaticiens ayant reçu une formation conséquente dans ce domaine de l'animation. Un effort supplémentaire est demandé généralement à l'utilisateur pour la manipulation d'une base de donnée importante qui regroupe des bibliothèques d'images.

3.2. Sur la production de "mouvements expressifs"

Une certaine désillusion semble s'emparer des utilisateurs quand à la réduction de coûts pour des productions d'une assez bonne qualité. L'ajout manuel de dessins intermédiaires, pour obtenir une meilleure expressivité risque de mettre en question l'utilisation de l'ordinateur-automate dans son assistance à l'animateur. On se propose actuellement d'intégrer des modèles biomécaniques dans le calcul des dessins intermédiaires pour rendre "naturels" les mouvements de personnages [15]. Un problème essentiel réside dans le système de production par image : le temps relativement long séparant la constitution des dessins, et leur restitution dynamique ,ne permet pas un expérimentation efficace des mouvements, c'est à dire d'une manière rapide et interactive.

CONCEPTS et MODELE PHYSIQUE

1. Objectifs du modèle Anima

Le système Anima essaie de se fixer pour but l'intégration du processus de création qui manque ou est mis de côté dans les procédés habituels d'animation utilisant l'ordinateur. Ces derniers sont surtout vus comme des outils de contrôle ou de documentation informatique de certaines techniques d'animation traditionnelle. Néanmoins ce système ne prétend pas se substituer à des systèmes éducatifs dans l'interprétation graphique de phénomènes physiques quant à l'animation produite, ni à court terme à des systèmes inspirés de l'animation conventionnelle utilisant le produit qu' est le film pour sa capacité à "raconter des histoires".

La maquette actuelle est une approche pour la réalisation de l'animation elle-même où le processus de création-exploration est privilégié. Les applications visées sont essentiellement de nature artistique. Le processus que l'on vient de citer comprend:

- la possibilité de créer des objets grâce à un langage adapté,
- la possibilité de les animer en simulant en temps réel leur comportement en réponse à des actions gestuelles exercées par un manipulateur,
- la possibilité de faire un apprentissage du mouvement, et des explorations faciles, grâce au temps réel et à une perception multisensorielle,
- la mise en place de certaines ressources, en particulier la "mémorisation de la manipulation" des objets, nécessaire à terme dans une phase compositionnelle.

Cette démarche de l'animation diffère essentiellement des autres par l'adoption des principes suivants :

. nous ne travaillons pas sur un système d'enchaînement d'images,

. la cohérence entre les images succéssives se trouve "codée" dans l'objet simulé,

. une séquence particulière d'images s'obtient en imposant à l'objet des actions extérieures qui jouent le rôle de conditions initiales pour la mise en mouvement d'un objet physique.

Dans cette étude on se bornera à l'expérimentation sur des objets à deux dimensions présentant des caractéristiques mécaniques inspireés de notre univers physique.

Nous introduirons en particulier dans ce chapitre le modèle mécanique conçu pour les objets, les modalités de manipulation de ces objets, et le procédé de visualisation.

2. Le modèle mécanique

Une description mécanique des objets, qui approche le mieux l'univers réel est sans doute souhaitable dans cette expérimentation. Cependant les contraintes sur la vitesse de simulation, sur la complexité de la maquette nous conduisent à :
- réduire la dimensionnalité de l'espace des objets simulés
- étudier des objets élastiques, c'est à dire ne comportant pas des articulations rigides.

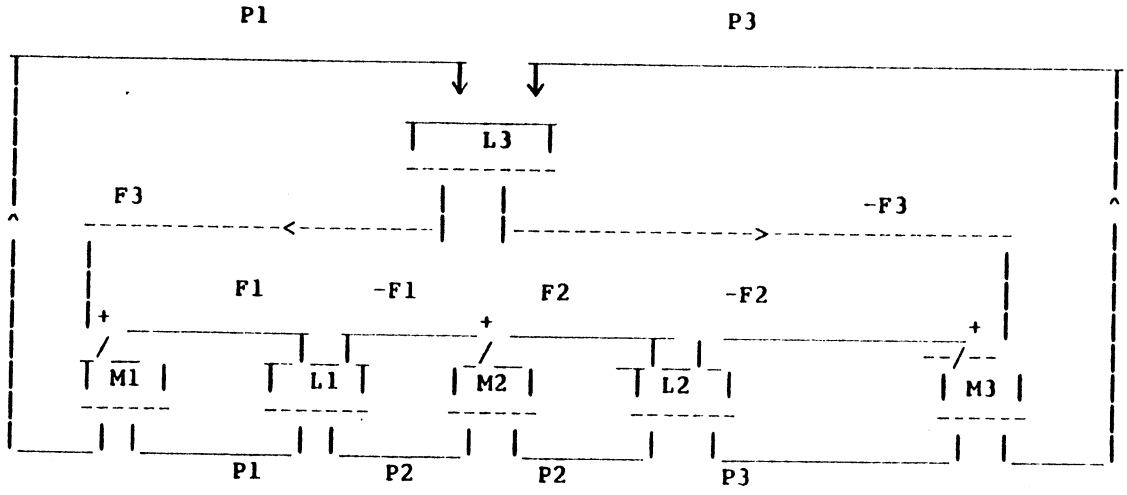
Dans ce cas les objets étudiés ici sont notamment constitués d'éléments matériels et d'éléments de liaison ressort-frottement qui vérifient les contraintes fondamentales suivantes :

(1) F ext = m * gamma
où F ext indique la résultante des forces appliquées sur une masse ponctuelle m d'accéleration gamma.
(2) F frot = z * dv
où F frot et - F frot indiquent les vecteurs de force de frottement appliqués respectivement aux points extrémités d'un élément frottement muni d'un coefficient de frottement z et m0s à la vitesse relative dv.
(3) F rap = k * dl
où F rap et - F rap indiquent les vecteurs de force de rappel appliqués respectivement aux points extrémités d'un élément ressort muni d'un coefficient de raideur k et ayant subi une élongation dl.

Les équations citées ci-dessus découlent des équations de la mécanique classique en milieu continu. Pour la simulation, le modèle choisi consiste à traiter l'objet comme un réseau discret faisant intervenir les masses ponctuelles sur les noeuds et des liaisons élastiques entre les noeuds. Dans ce cas l'équation d'équilibre de chaque masse (M) permet de calculer sa position (P) à chaque instant, et l'équation des forces (F) sur la liaison (L) permet de calculer le module des forces appliquées sur ses attaches, c'est à dire aux masses sur les 2 extrémités. Par ailleurs des études approfondies ont été faites [43] sur la stabilité du réseau ainsi constitué en tant que dicrétisation

du milieu continu.

Nous montrons ici la "circulation" des variables forces et positions sur un réseau à trois noeuds faisant intervenir une liaison entre chaque noeud.



La réduction sur la dimensionnalité ne semble pas nuire à une "pertinence" du mouvement, par ailleurs les caractéristiques physiques attribués aux éléments de l'objet (masse, constantes raideur et frottement) répondent bien à des critères de perception visuelle c'est à dire provoquent des sensations visuelles faciles à appréhender. Des expérimentations simples mettant en oeuvre les équations précédentes sur des élements 1D et pseudo 2D ont été faites et présentent des résultats surprenants quand à la richesse du mouvement [46]. La structure du programme utilisé a cet effet est présenté au chapitre (IV- 4.).

Nous établissons ici la liste des modules mécaniques mis en oeuvre dans la description de l'objet et accessible aux commandes de l'utilisateur. Du côté utilisateur, le réseau mécanique discret représentant l'objet est formé par une combinaison de ces modules. D'autre part la notion de module recouvre un aspect algorithmique, c'est à dire qu'elle représente l'algorithme relatif à la résolution des équations régissant le comportement d'un ensemble d'éléments de

base du réseau mécanique qui sont des masses ponctuelles, des ressorts-frottements, et des points fixes. Pour les modules élémentaires cités ci-dessus ces ensembles sont à un élément. La définition d'un certain nombre de ces modules découle directement de l'étude faite sur les objets de dimension mentionnée au paragraphe précédent (1D et pseudo 2D). L'étude actuelle a permis en partie de cerner les problèmes propres aux modules 2D, et à des modules plus évolués (ex. les modules primaires, les modules de liaison conditionnelle). Cette étude a permis aussi de trouver un meilleur formalisme, d'une part dans le langage utilisateur pour la définition des modules qui constituent l'objet à créer, d'autre part dans la communication interne entre les modules dans la phase de simulation.

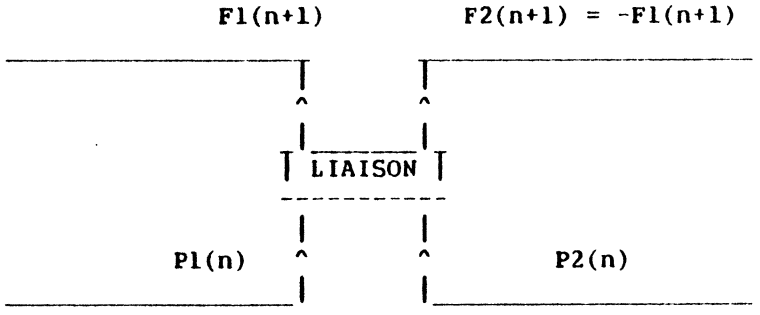
Nous définissons ici les modules soit à partir des équations explicites mises en oeuvre dans les algorithmes, soit pour simplifier, à partir des variables d'entrée et de sortie des algorithmes.

2.1. Modules élémentaires

a) Module masse
L'équation discrétisée qui régit la masse est :
$P(n) = F(n-1)/M + 2.P(n-1) - P(n-2)$
où P(n) est la position(x,y) à l'instant de calcul n.
Elle résulte de l'approximation $G(n) = V(n)-V(n-1)$ et
$V(n) = P(n)-P(n-1)$
qui consiste à approximer l'intégration de l'équation (1) en passant à l'équation aux différences. Nous pouvons retrouver ce type d'intégration pour le reste des équations.

b) Module ressort-frottement
Les équations suivantes décrivent les forces appliquées aux éléments matériels attachés aux extrémités de l'élément ressort-frottement.
$F(n) = FR(n) + FF(n)$ où FR et FF sont les composantes des forces frottement et force de répulsion.
Pour la composante en x on obtient les équations suivantes :
soit lg = distance(P1(n),P2(n))
$FRx(n+1) = [k * (lg - lginit)] * [X2(n) - X1(n)] / lg$
soit Produit_scal = $[(X2(n)-X2(n-1) - (X1(n)-X1(n-1))*(X2(n)-X1(n)) + (Y2(n)-Y2(n-1)- (Y1(n) - Y1(n-1))* (Y2(n)-Y1(n))]$
$FFx(n+1) = z * (X2(n)-X1(n)) * Lg^{**}2 *$ Produit_scal
Lg indique ici la longueur courante du ressort et lginit sa longueur initiale. La composante y se calcule de la même manière.

Les forces appliquées aux deux attaches sont respectivement F(n) et -F(n) à l'instant n.



c) Module sol
Ce module délivre une position constante à tout instant quelles que soient les forces appliquées.
P(n) = P0

2.2. Modules élémentaires dégénérés.

Ces modules sont assimilés à des liaisons. Ils fournissent des forces indépendamment des positions de leurs attaches.
a) Module attraction
On s'intéresse notamment à l'attraction terrestre :
F(n+1) = M.(P(n)-2.P(n-1)+P(n-2))
b) Module moteur
Ce module fournit un couple de forces constant sur ses attaches.

2.3. Les modules primaires

Ce sont des modules moins élémentaires qui ont été introduits pour augmenter l'efficacité au niveau de la description et au niveau de la simulation. Au niveau de la description l'utilisateur garde des possibilités d'accès aux éléments mis en jeu par le module.
Nous noterons les modules suivants :

a) Module liaisons au sol
Le nombre d'éléments de liaisons peut être de 1 ou de 2 ou de 3. Chacun de ces éléments a une extrémité attachée à un élément sol.
b) Module corde
Il est caractérisé par une identité sur les éléments masses qui le constituent et sur les éléments liaisons intermasses.
Trois couples (raideur,frottement) sont attribués aux deux liaisons aux extrémités et à la liaison intermasse. Les 2 éléments extrémités sont des éléments sol.

c) Module étoile
Il est caractérisé par une identité des éléments ressort-frottement qui le constituent . Une extrémité de chacun de ces éléments est reliée à un même élément masse .

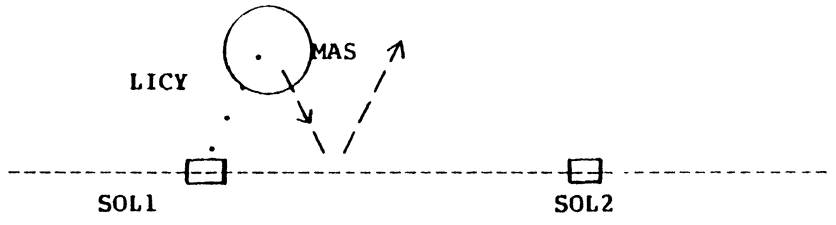
2.4. Les liaisons conditionnelles.

La simulation des situations de chocs ou de contact entre les objets s'avère être coûteuse si on fait une évaluation systématique des conditions pouvant impliquer ces situations. C'est pour cette raison que des tests de contact n'ont pas été abordé jusqu'à présent au niveau des modules. Néanmoins, la nécessité d'une telle simulation pourra être analysée statiquement à partir des conditions initiales de la mécanique (recherche de mouvements périodiques, trajectoires parallèles,..) dans les cas où les trajectoires des objets ne sont pas influencés par des actions de l'opérateur. Dans l'ensemble des cas, une telle analyse ne nous semble pas faisable avec un temps de réponse raisonnable compte tenu du nombre d'équations et de variables rentrant en jeu, et elle n'a pas été intégré au niveau du modèle.

La définition de la liaison conditionnelle permet de simuler une situation de contact entre des éléments de l'objet, et de définir une équation particulière régissant ces éléments pendant cette situation. Cette équation indique l'établissement d'une liaison ressort-frottement (idem 2.1 b) que nous appelons conditionnelle entre deux éléments matériels p1 et p2 en contact c'est à dire qui vérifient la condition d(p1,p2)<seuil. D indique la distance entre les deux éléments matériels .

Nous introduisons ici deux types de liaison qui sont :
- la liaison conditionnelle à deux dimensions. L'équation de la liaison définie au (2.3.b) est alors intégrée au calcul, pendant le temps où la condition est vérifiée. Lginit prend ici la valeur du seuil.
- la pseudo-liaison conditionnelle à une dimension . L'équation d' un ressort frottement à un degré de liberté est intégrée dans le calcul. Cette équation est plus simple car elle ne met pas en jeu le calcul du terme en racine carrée dans la distance entre les points. Pour une liaison en x, le vecteur force généré est le suivant :
fx = k*dx + zdv avec
dv = (X2(n)-X2(n-1)-(X1(n)-X1(n-1))
dx = (X2(n)-X1(n))-(X2(0)-X1(0))
où X1(n) et X2(n) représentent les positions courantes des 2 éléments matériels.
Ce dernier type de liaison permet de simuler facilement un choc entre un élément matériel et une ligne complètement rigide représentée par deux modules Sol dont les points sont situés sur un même axe vertical ou horizontal, entrainant une réflexion totale du vecteur vitesse

incident.



2.5. Accès à l'objet mécanique

2.5.1. Transducteur gestuel, Canal

Dans ce paragraphe on s'intéresse à la définition des modules qui traitent la communication gestuelle entre le manipulateur et les objets en phase d'expérimentation. Cette communication pose le probleme de la capture du geste ainsi que de sa structuration. Pour cela nous reprenons brièvement. deux concepts qui ont été largement introduits dans les rapports de recherche de l' équipe qui sont le transducteur gestuel et le canal. Des informations plus fournies peuvent être retrouvées dans [45,47].

Le transducteur gestuel est un dispositif matériel qui permet de prendre en compte une activité gestuelle. Il transforme des informations de force, d'espace et de temps du geste en impulsions éléctriques codées qui seront décodés au niveau du calculateur hôte. Les informations sont ensuite transmises à l' objet mécanique simulé. Dans le cas d'un transducteur rétroactif, le dispositif est capable de rendre à l'opérateur la réaction de l'objet mécanique, toujours par le canal gestuel, en réponse à une manipulation première. Au niveau du calculateur ce mécanisme de transmission sera pris en compte par les modules d'entrée que nous avons décrit plus loin. Un ou plusieurs transducteurs gestuels peuvent être mis en jeu simultanément. Un prototype de transducteur rétroactif à un degré de liberté a été réalisé au sein de l'equipe [12] et une étude sur un transducteur rétroactif à plusieurs degrés de liberté est en cours.

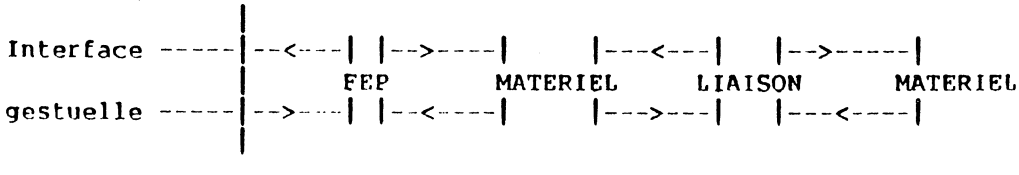
Le concept de canal permet de définir une typologie sur le geste effectué. Dans une activité gestuelle nous distinguerons notamment 2 composantes dans le geste fourni:
- le geste d'excitation : il fournit l'énergie à l'objet et il est sujet à une rétroaction.
- le geste de modulation : il fournit une énergie minimum qui permet de faire une modulation sur les caractéristiques mécaniques de l'objet et ne participe pas directement à l'énergie de déplacement de l'objet (ex. le glissement des doigts sur un manche qui modifie la tension des

cordes sans les exciter). Nous supposons que dans ce geste le rôle de la rétroaction est négligeable.
Nous entendons fixer une limitation à la complexité de l'activité gestuelle. Pour nous celle ci peut mettre en oeuvre au maximum 2 canaux, chaque canal regroupe les voies suivantes:
- 1 voie d'excitation qui supporte le geste d'excitation. Nous associons un nombre de degré de liberté à cette voie qui est identique à celui du transducteur utilisé.
- n voies de modulation qui supportent le geste de modulation. Nous associons un dégré de liberté egal à 1 à la voie de modulation.

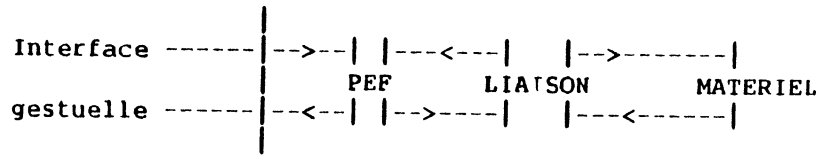
Au niveau du calculateur, la voie d'excitation de degré d sera prise en charge par d voies analogiques-numériques (pour l'entrée), et d convertisseurs numériques-analogiques (pour le retour). La voie de modulation est prise en charge par un convertisseur analogique numérique. Les convertisseurs se trouvent sur une carte interface du calculateur, permettant la connexion avec les transducteurs.

2.5.2. Les modules d'entrée

Ils prennent en charge le traitement des voies d'entrée au niveau de la boucle de simulation. Nous integrons le module de traitement de la voie d'excitation dans le formalisme qui relie un module matériel à un module de liaison. Dans ce cas le geste d'excitation peut être vu ,soit comme un geste appliquant une entrée en force et recevant une position en retour, soit comme une entrée en position et recevant une force en retour. Dans le premier cas le module qui prend en charge la voie (module FEP) est assimilé à un module liaison, et dans le second cas (module PEF) il est assimilé à un module matériel selon le formalisme général adopté pour la "circulation" des variables force et position. Cette similitude sera donc reflétée au niveau du langage de commande de l'utilisateur, au niveau de la représentation interne de ces modules et au niveau de leur simulation.
On peut ainsi schématiser la chaine Entrée geste-objet dans la cas d'un objet de structure linéaire comme-ci :
1er cas)



2eme cas)



La voie de modulation ne suit pas ce formalisme car le geste modulant ne contribue pas à un échange réel d'énergie avec l'objet ayant ses caractéristiques modulés. On considère néanmoins que la voie de modulation fournit une entrée en position ou une entrée en force. Les modules PEM et FEM se chargent de traiter la voie dans l'un ou l'autre des 2 cas.

Nous avons introduit ici une interface gestuelle qui permet de faire le "conditionnement" des informations issues de la capture du geste pour qu'elles soient assimilables directement par les modules d'entrée. Les informations que génèrent cette interface sont donc des variables force ou position.

2.6. Le module Contrôle de paramètre (CEP)

Nous avons introduit le module CEP pour plus de souplesse en complément des modules PEM et FEM pour faire la modulation effective des caractéristiques mécaniques. Il nécessite la création préalable d'un module PEM ou FEM dont il utilise une variable de sortie pour faire varier en cours d'expérimentation les caractéristiques mécaniques de l'objet suivant une fonction linéaire.

Remarque

Une structuration par tableaux des informations englobant les concepts introduits a été dévelopée au niveau des outils utilisateurs pour faciliter la définition des activités gestuelles. Cette définition a pour but :
- la création des modules d'entrée,
- la spécification de l'interface gestuelle nécessaire à l'expérimentation.

2.7. Modules de visualisation

Ces modules contrôlent la génération graphique, représentant des variables ou des comportements de variables caractérisant l'objet en

expérimentation. Les modules de visualisation et le procédé d'affichage adopté définissent ce que nous appelons le transducteur visuel. Nous nous interessons notamment à la représentation des variables positions des éléments matériels. Les 2 modules principaux sont :
- VECTEUR qui affiche le vecteur représentant la distance séparant 2 éléments matériels,
- POINT qui affiche le point symbolisant la position d'un élément matériel.

Le transducteur visuel permet donc de manière générale de faire une représentation géométrique (une "spatialisation") à partir du domaine mécanique.

Pour respecter les contraintes de vitesse d'une expérimentation temps réel nous nous sommes intéressés dans un premier temps à l'affichage d'éléments graphiques simples qui correspondent à l'affichage d'un squelette géométrique. Nous trouvons judicieux dans ce cas l'utilisation d'un écran à balayage cavalier, qui offre généralement une bonne résolution de l'image au trait. Cette situation doit aboutir à l'ajout d'un habillage géometrique du squelette géométrique en gardant le même procédé de visualisation et une même classe de complexité de modules.

Au niveau de l'affichage de l'image finale, deux types de traitement peuvent être considérés :

- un premier type faisant intervenir un processeur graphique qui gère et interprète les segments d'affichage. Dans ce cas, le processeur central qui fait le traitement des modules de visualisation envoie à ce dernier les commandes globales qui résultent de la "spatialisation" (par adjonction ou suppression de segments d'affichage),
- un second type faisant intervenir un simple générateur de vecteur en l'absence d'un processeur graphique. Dans ce cas les fonctions du processeur graphique est pris en charge par le processeur central. Ce type de traitement a été retenu pour la réalisation actuelle.

A moyen terme on envisage une génération d'images complexes, faisant intervenir une composition d'objets complexes où il sera nécessaire d'introduire un écran à balayage de trame, couplé à un processeur graphique du type synthétiseur d'images. On affectera essentiellement à ce processeur le remplissage des zones. Une animation temps réel des images ne sera néanmoins pas possible compte tenu du temps de simulation de tels objets et du temps de génération actuelle d'une image pleine. Les traitements spécifiques à ce dernier type d'images ne sont pas abordés dans ce travail.

LE SYSTEME ANIMA

1. Le cahier de charges

Les spécifications de ce cahier sont d'ordre assez général car elles prennent en considération un maximum de possibilités qu'offre le modèle théorique. Nos choix seront sûrement influencés par la configuration matérielle disponible et les contraintes de temps pour aboutir à un système utilisable à très court terme. Nous exprimons à cette occasion le désir d'aboutir à un système où l'utilisateur peut accéder de manière interactive à toutes les fonctions, comparativement à des "systèmes" ayant pour but la production d' images animées, où cette interaction est difficile à atteindre en raison des contraintes temporelles dûes essentiellement aux temps de transformation des images. Il s'agit dans plusieurs cas d'une association de ressources diverses plutôt que de "système" qui permet d'aboutir à une cohérence sur les éléments au niveau du produit final qu' est le film.

Nous nous soucierons dans la mesure du possible de garder un système ouvert à des extensions, sur les fonctions où il n'y a pas de limitation de principe (complément de représentation, modules de simulation ..).

Notre choix porte sur une articulation du système autour de trois tâches essentielles :

a- la description des objets
Cette description comprend :
. la description mécanique et géométrique,
. la description des sorties visuelles,
. la description des fonctions de manipulation et de modulation gestuelle,
. la fonction de mémorisation sur fichiers des objets.

Les questions que nous aurons d'abord à aborder au niveau de cette tâche concerne :
- le choix de représentation à adopter,
- le choix des commandes de base de l'utilisateur.

Ceci nous amène à prendre en considération quelques outils de base permettant d'introduire le logiciel graphique ANIGRAPH qui a servi en partie à répondre à la constitution de ces outils.
Nous présentons ensuite :

- des outils plus évolués (au niveau de la représentation externe et du traitement des commandes) qui utilisent la représentation adoptée et les commandes de base. Quelques aspects théoriques sont évoqués notamment au niveau des outils de description de l'activité gestuelle pour contribuer à une meilleure structuration de l'information au niveau de l'utilisateur,

- la structuration générale des commandes utilisateur,
- la représentation interne des objets,
- l'organisation générale du programme de description des objets.

b- la définition d'un environnement d'expérimentation
Cette tâche a nécessité :

1) l'étude de l'ensemble des processus en jeu
Cette étude sera préfacée par une présentation générale des processus. Nous entamons par la suite une étude détaillée de ces processus dont une partie a fait l'objet d'une implantation. Cette étude comprend notamment l'étude des problèmes de synchronisation, les problèmes de communication dûs à l'utilisation de deux processeurs, l'interfaçage des processus avec les organes matériels nécessaires à l'environnement (transducteur, mémoire de masse, traceur,...).
2) la description des modes d'utilisation :
.la définition du mode d'accès à l'objet mécanique (utilisation de transducteurs ou lecture d'échantillons gestuels sur fichier)
.la définition du mode de visualisation (sortie des échantillons visuels sur écran rapide, ou autre organe plus lent tel que disque,traceur...)
.la définition des points de vue d'expérimentation (espace simulé, espace visualisé, vitesse d'affichage et de calcul,vitesse d'échantillonnage du geste de manipulation).

c- implantation des modules de simulation
Cette tâche correspond à l'implantation sur machine des modules de simulation, dans des conditions qui favorisent au mieux le temps réel. Dans un premier temps il s'agira d'aprofondir les possibilités qu'offre le processeur AP-120B, ainsi que de son fonctionnement avec le calculateur LSI 11/02 en tant que calculateur hôte. Nous aborderons par la suite l'étude du contexte nécessaire au processus de simulation notamment la vectorisation des données et enfin de la procédure implantée du côté hôte pour la génération de ce contexte.

2. Description des objets

2.1. Présentation

On aborde ici deux thèmes généraux qui sont la représentation d'objets par la machine et dialogue opérateur-ordinateur. Cette représentation recouvre deux aspects qui sont la représentation interne et la représentation externe (ou représentation à l'utilisateur). Par ailleurs, le type d'objets

représentables peut être très variable car il peut s'agir d'objets constitués d'éléments de la mécanique dont il est question ici, mais peut être aussi un fichier ou un programme ou autre. Généralement le dialogue et la représentation externe varient suivant le type d'objet traité et la classe d'utilisateur concerné (novice, averti, expert). Nous pensons qu'il n'existe pas vraiment de doctrine dans ces domaines où souvent les solutions diffèrent selon les applications. Nous n'essayons pas d'apporter ici une solution formelle mais nous évoquons quelques principes de base [ ] sur lesquels nous allons nous inspirer pour l'application actuelle. Sur la représentation externe des objets nous citons les principes suivants:

- l' utilisation d'une représentation analogique (graphique), avec la possibilité de coder des valeurs associées aux objets et l'identité des objets est complémentaire à l'utilisation d'une représentation alphanumérique. Cela permet d'avoir une perception globale et spatiale des informations de proportionnalité entre les éléments de l'objet.

- la complexité de la représentation ne doit pas dépasser un certain seuil en terme de quantité d'observation à percevoir. Une gestion adéquate de l'écran est dans ce cas nécessaire, ainsi que des fonctions de filtrage des informations. Une représentation trop simple est toutefois à éviter.

- les modifications de représentation doivent répondre à des critères d'expérience et de prévisibilité humaines. Cela évitera de nuire à l'efficacité du dialogue et à la coordination des actions de l'utilisateur.

Sur le thème "dialogue homme-machine", on fera apparaitre notamment le rôle de l'interface utilisateur ("user interface") sur lequel de nombreux travaux d'analyse conduisent à dire qu'il s'agit moins d'intelligence artificielle que d'un souci de concevoir un art de dialogue]. Nous pouvons citer à cet égard la nécessité de respecter des contraintes informelles sur les temps de réponse, la récupération en cas d'erreur, la clarté des messages et l'ergonomie des périphériques utilisés.

Des niveaux de formalisation ont néanmoins été atteints, tels que l'implantation de primitives supportant des langages d'entrée de commandes[25], ou la conception de langage interactif tel que Smalltalk[51]. Notre étude dans ces deux domaines sera grandement influencée par les rapports des groupes d'étude sur l'interaction homme-machine édités par Guedj[51], les travaux de Lucas[30], et une synthèse sur les outils de base sur la communication graphique (logiciels GKS, Gri-gri, Clovis)[27,28,32]

2.2. Présentation des éléments de dialogue pour ANIMA

2.2.1. Type de représentation et type de commande adoptés

Pour le système ANIMA nous nous intéresserons à une représentation graphique liée à une construction interactive des objets appelée plus communément "graphisme de construction" [. Cette fabrication permet d'agir en partie sur la représentation au moyen de commandes graphiques (suppression d'un élément par pointage sur sa représentation, création d'un élément par désignation d'un point dans une zone d'écran servant à sa représentation,...). La représentation actuelle fait apparaitre notamment deux types d'informations :
a) informations "symboliques" : représentation de l'activité gestuelle et relation de l'objet mécanique avec l'expérimentateur;
b) informations "figuratives" : définition spatiale et quantitative de l'univers physique simulé.

Nous écartons volontairement ici l'utilisation d'un éditeur lignes (une description textuelle des objets) qui serait fastidieux à réaliser, dont le texte nécessiterait une analyse syntaxique et sémantique, et rendrait difficile une correction interactive en cas d'erreur. En effet notre choix porte sur l'elaboration d'un ensemble de commandes interactives contrôlées individuellement pour faciliter le travail de l'utilisateur. Du côté programme cela permet une mise à jour "au vol" des données et d'éviter des procédures lourdes pour la récupération d'erreur.

2.2.2. Matériel utilisé pour le dialogue

L'étude sur les outils présentés ici a été influencée par l'environnement matériel disponible dans la mesure où le travail actuel a beaucoup mis l'accent sur l'aspect réalisation. Une partie même de ces outils assure l'interface avec le matériel qui comprend :

a) une console graphique monochrome SECAPA avec contrôleur graphique et balayage video d'une mémoire d'entretien 512*512 points,
b) une tablette graphique munie d'une souris, reliée à la console et permettant l'acquisition de coordonnées écran à partir du réticule,
c) un capteur de force tridimensionnel monté sur la souris, permettant l'acquisition de valeurs de force, pour une rentrée analogique de paramètres lors du dialogue. La conversion A/D se fait sur la même carte de convertisseurs que celle utilisée pour les transducteurs nécessaires à la modulation des paramètres et la manipulation en phase de simulation.

Nous désignons ici et pour la suite sous le terme général de paramètres les valeurs affectées aux caractéristiques des éléments mécaniques (ex. coefficient de raideur ou de frottement d'un ressort) que l'utilisateur fournit dans la description des objets. Une liste détaillée de ces paramètres est faite dans le chapitre sur l'implantation des modules de simulation.

2.2.3. Les opérations de base pour l'utilisateur

Le dialogue est basé sur une forme simple de conception assistée dont les opérations de base se résument à:

- pointer un point d'écran à partir du réticule
- entrer une valeur alphabétique ou numérique sur une zone bien définie de l'écran,
- agir sur le capteur de force.

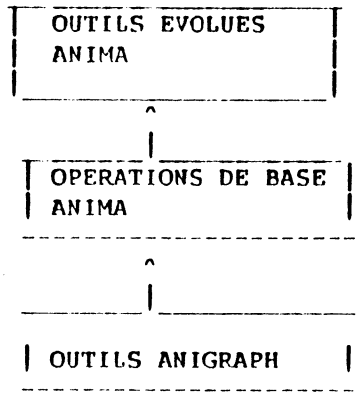
La premiere opération permet d'effectuer les fonctions suivantes :

- entrée d'une position,
- identification d'une fonction de menu,
- identification d'une case de matrice . Nous désignons ici par matrice une représentation graphique sous forme de tableau à double entrée permettant de visualiser et d'acquérir des valeurs dans un ordre indiqué par des "légendes" apparaissant sur la première ligne et la première colonne,
- identification d'une entité graphique.

La deuxième opération permet de rentrer l'identification d'une entité affichée sur l'écran, ou l'affectation de paramètres à un élément de l'objet, affiché sur l'écran, ou la mise à jour de certaines données non représentées graphiquement (points de vue de simulation tels que temps, unités,...). Ces valeurs sont entrées dans une case de matrice, ou à la position du curseur à la suite de l' affichage d'un message pour permettre un repérage facile pour l'utilisateur.

La troisième opération permet la mise à jour interactive d'un paramètre d'un élément de l'objet. Cette mise à jour est contrôlée à partir des modifications sur l'état de représentation courant de l'élément, directement perçus en fonction de la pression sur les capteurs. Une primitive évoluée permettant de réaliser cette opération est décrite au niveau du chapître sur les outils pour l'interface utilisateur. Les primitives du logiciel ANIGRAPH que nous abordons maintenant ont permis de réaliser une partie de cette opération et le reste des opérations de base. L'ensemble de ces opérations est par ailleurs utilisé

dans les outils décrits dans le chapitre cité ci-dessus.
Nous pouvons illustrer cette "hiérarchisation" des outils comme suit:



2.3. Le logiciel de base de dialogue ANIGRAPH

2.3.1. Introduction

Ce logiciel de base a été concu et réalisé pour répondre aux préoccupations suivantes :
. modularité et indépendance entre l'application et les fonctions de représentation et de dialogue
. interfaçage avec le matériel.

A priori il était intéressant de partir d' outils existants que nous avons cités plus haut (2.1) et de les adapter à notre machine actuelle. Nous ne faisons pas ici un résumé de ces logiciels dont les descriptions profitent de larges diffusions[]. Malgré un effort de normalisation dans la spécification de ces outils nous pensons que faire notre propre réalisation a été plus profitable vu les contraintes posées à l'heure actuelle par :
a) l'implantation sur une petite machine à 32K mots munie du système RT11 SJ. Ces contraintes comprennent notamment la portabilité des programmes, la taille des programmes, le langage de programmation, l'interface avec le système et le matériel.
b) un souci d'efficacité qui demande une structuration des données graphiques et un langage de primitives adaptés au type d'application traité afin d'éviter une structuration de données trop lourde et un langage trop riche.

Cette implantation a été directement inspirée de ces outils qui sont basés essentiellement sur les concepts suivants:
. abandon de la notion de point courant,

. étude d'un ensemble de primitives cohérentes plutôt que d'une collection de primitives indépendantes .

Ce logiciel a été écrit en Pascal et nous pensons qu'il ne pose pas de problème de portabilité sur une machine LSI 11 ou PDP 11 dans sa structuration actuelle. Une version a été installée sur le VAX 730 où les sorties peuvent être dirigées sur une Ramtek , et sur une table traçante.

2.3.2. Structure de données

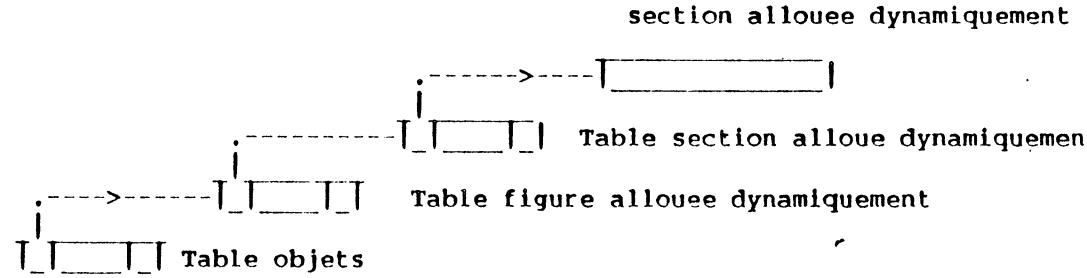
Les primitives du logiciel de base permettent de définir et de manipuler :
. des objets graphiques
. des outils de dialogue.
Ce paragraphe traite essentiellement des données qui représentent les objets graphiques.

2.3.2.1. Nécessité de la structuration

Une certaine structuration des données nous semble indispensable pour répondre à une structuration des objets vue du côté application. Cette attitude a été adoptée dans Gri-gri et Clovis où on a la possibilité de définir des objets graphiques représenté par un arbre de profondeur 2 pour le premier et de profondeur quelconque pour le second. Par ailleurs la notion de classe apparaît dans Gri-gri en introduisant un attribut (numéro de corrélation) permettant de regrouper les feuilles ayant le même attribut. La notion d'arborescence ou de classe permet de définir des primitives capables :
- d'affecter des attributs sur les éléments d'une classe donnée ou sur les noeuds d'un sous-arbre donné.
- d'effectuer des opérations pour une classe ou un noeud. La structure d'arbre permet par exemple l'affectation d'une priorité sur l'affichage, relative aux parcours possibles des noeuds.

Nous avons adopté dans Anigraph une structuration arborescente à

quatre niveaux des objets graphiques que nous schematisons ici : ensemble-objets-graphiques -> objet-graphique -> figure -> section



Les feuilles de l'arbre contiennent les informations élémentaires pour la représentation graphique, le reste des noeuds contient des pointeurs. La mémoire nécessaire à chaque niveau est allouée dynamiquement, suivant les besoins de l'application c'est à dire au fur et à mesure de la création de nouvelles figures et sections nécessaires à la représentation.

La structuration actuelle a été adoptée pour faciliter au mieux la correspondance avec celle des objets de l'application :

. Le premier niveau permet de coder une scène affichée qui peut inclure une représentation simultanée de plusieurs objets de l'application. L'objet considéré ici est celui dont la description est prise en charge par première tâche du système indiquée dans le cahier de charges,
. l'objet-graphique permet de représenter l'objet,
. la section permet de représenter un module élémentaire ou une information symbolique (2.2.1) au moyen d'un symbole graphique.
. la figure permet de représenter soit un module primaire, soit un ensemble de modules élémentaires et de symboles graphiques.

Les primitives graphiques sont applicables à chaque noeud de l'arbre (affichage, effacement, gestion affichage des marqueurs...). Nous n'avons pas d'opérations de transformation géométrique sur les objets-graphiques, car ils ne sont pas nécessaire actuellement au dialogue dans de notre application. D'autre part nous n'introduisons pas ici de "fichier graphique" (mémorisation des objets graphiques) car l'ensemble des données graphiques est générée "rapidement" à partir des informations analytiques de l'application.

2.3.2.2. Description de la section graphique

Nous décrivons ici les informations contenues dans une section regroupant deux attributs qui sont le contenu et l'ensemble des caractéristiques d'affichage. Nous ne revoyons pas en détail ici des notions "classiques" (constituants du contenu, clôture, fenêtre,...) qu'on peut retrouver dans [34]

1- Le contenu
La section comprend soit un buffer de caractères, soit un buffer de coordonnées dans un plan orthonormé. Il représente un texte ou un dessin à afficher dans une clôture de l'écran.

2- Les caractéristiques d'affichage
Nous retrouvons ici :

a) la clôture (zone rectangulaire d'affichage en coordonnés écran) et la fenêtre (zone rectangulaire dans le plan orthonormé incluant les coordonnées à prendre en compte) pour l'affichage du contenu.
Nous utiliserons pour cela l'algorithme de clipping[pour les vecteurs, et on affichera la section de texte en mode rouleau dans la clôture d'écran associée.

b) le mode d'interprétation du contenu (section de texte, points, vecteurs disjoints, vecteurs joints),

c) la description des attributs d'affichage (taille des caractères, inclinaison, texture des vecteurs : pointillés, tiretés, mixtes),

d) la description d'un encombrement géographique pour la section dans le système de coordonnées adopté. Cela sera utilisé par un algorithme d'identification de la section à partir d'un pointage sur écran par le réticule ou un crayon lumineux. Dans le cas d'une section de texte on adoptera la clôture d'affichage, sinon c'est un champ rectangulaire,

e) une variable pointeur sur un marqueur qui est associé à la section.
Le marqueur est une chaine de quatre caractères affichable et sa position d'affichage est relative à l'encombrement décrit précédemment. Dans l'application actuelle le marqueur est utilisé pour identifier un module élémentaire ou un module primaire représenté sur l'écran. Chaque marqueur est une entrée dans un tableau de marqueurs. L'adjonction ou la suppression d'un marqueur se fait à partir de primitives ANIGRAPH.

f) un pointeur sur une file d'affichage de la section.
Cette file contient la succession de codes à envoyer au contrôleur graphique pour l'affichage ou l'effacement de la section. La liste est créée lors d'un appel à une primitive d'affichage de section. L'effacement consiste à envoyer les mêmes codes précédés d'un code d'effacement. Les files permettent de mémoriser sous forme de listes linéaires l'ensemble des sections affichées sur l'écran. Cette structuration en effet n'apparaît pas au niveau de la mémoire d'entretien qui est un tableau de pixels. La liste est alors nécessaire pour :
- un effacement partiel ou total,
- un réaffichage partiel ou total notamment de sections présentant des "trous" dûs aux effacements partiels. Cela est fait sans regénérer la liste des codes graphiques et sans faire un parcours des sous-arbres qui contiennent les sections.

2.3.3. Identification d'une section après une opération de pointage

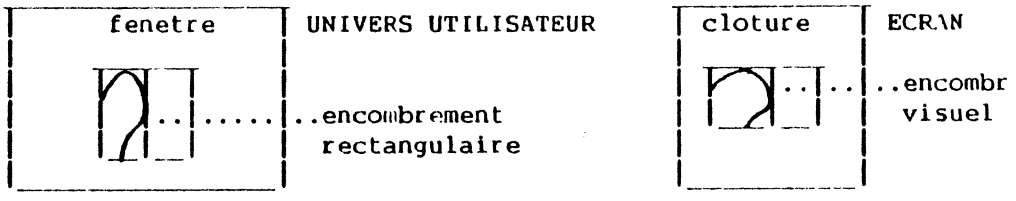
Recherche par intersections maximales sur les encombrements visuels

L'algorithme utilisé se veut adapté à des problèmes d'identification relative à des objets ayant une silhouette graphiquement localisée. A propos du problème d'identification les algorithmes peuvent être multiples et complexes car ils ils peuvent répondre à des critères très divers : type d'objets graphiques manipulés, précision de pointage demandée, temps de réponse,...

Ici nous faisons l'identification d'une section (de l'objet graphique) à partir d'une silhouette qui lui est associée. Nous avons choisi une forme simple qui est le rectangle d'encadrement de la section (caractéristique d'affichage d- appelé encombrement géographique). Les conditions suivantes nous ont conduit à adopter ce choix :

- les dessins représentant les éléments masse et ressort rendent acceptable l'approximation par la forme rectangulaire. Le cas le plus défavorable est celui du ressort incliné à 45 degrés par rapport à l'axe vertical ou horizontal,
- la reconnaissance (ou non) est presque instantanée. L'identification de la section permet de retrouver la figure ainsi que l'objet graphique.
- l'identification est suivie d'une commande de validation par l'opérateur.

Pour la suite de l'explication où nous présentons l'algorithme nous utiliserons le terme d'encombrement qui indique l'encombrement visuel c'est à dire celui de la partie de la section visible sur l'écran mais non celui dans l'univers utilisateur (repère utilisé pour les coordonnées). Celui-ci est mémorisé comme une caractéristique statique de l'objet-graphique, alors que l'encombrement visuel est calculé en fonction de ce dernier et en fonction de la clôture et de la fenêtre d'affichage.



Après pointage, les critères d'identification sont les suivants :
1) la section identifiée est la seule dont l'encombrement contient le point,
2) si il y a conflit c'est à dire que le point appartient à plusieurs encombrements on procède de la manière suivante :

a) on adopte une fenêtre rectangulaire à partir du point,
b) on fait l'intersection de la fenêtre et des encombrements en conflit. On sauvegarde les encombrements ayant la surface (rectangulaire) maximale d'intersection avec la fenêtre. L'opération est répétée à partir du point b) avec une fenêtre agrandie, jusqu'à obtention d'un encombrement unique,
c) la section identifiée est celle associée à cet encombrement final.

Il en découle quelques techniques simples de pointage pour optimiser la recherche :
- pointer dans un encombrement non partagé
- le cas échéant, répérer puis pointer le centre de l'encombrement de l'élément de l'objet à identifier.
La difficulté est ici d'apprécier les encombrements visuels.

En plus d'une relative simplicité de réalisation, l'algorithme présente des cas avantageux : convergence rapide, pointage imprécis accepté (ce qui n'est pas toujours le cas pour une recherche de proximité sur les vecteurs[]). Des cas limites peuvent se présenter :
1) des encombrements se superposent. Dans ce cas la boucle doit être contrôlée car il n'y a pas de convergence. Un des objets est

choisi arbitrairement.
2) un encombrement est inclus dans un autre. L'objet associé à l'encombrement englobant est identifié.
Un raffinement de l'algorithme peut consister à choisir un autre type d'encombrement (autre forme polygonale ou chaine d'encombrements rectangulaires), mais n'exclut pas les cas de conflit (ex. inclusion). Nous pensons que dans notre cas la mémorisation et le calcul de tels encombrements seraient onéreux en raison des formes d'objets choisis et de la possibilité de faire une désignation alphanumérique.

2.3.4. Les outils de dialogue

Nous associons à un outil de dialogue une structure de donnée et des primitives. Nous avons adopté les 2 outils principaux suivants : le menu et la matrice. L'utilisation du menu est familière, elle permet de proposer un certain nombre de fonctions à exécuter dont un choix est établi à partir d'un pointage. La caractéristique associé à un menu est un ensemble de noms (d'objets ou de fonctions) qui seront disposés en lignes consécutives lors de l'affichage du menu et la position d'affichage.

La matrice permet de représenter graphiquement un tableau à double entrée, présentant des "légendes" (une légende est mémorisée comme une chaîne de 8 caractères) sur la premiere ligne et sur la première colonne. Des variables entières, réelles ou chaine de caractères peuvent être édités.

Les caractéristiques associés à une matrice sont la position d'affichage et les noms des légendes. Le contenu de la matrice n'est pas mémorisé au niveau du logiciel graphique mais uniquement au niveau du logiciel d'application pour éviter les redondances. Ceci n'est pas gènant car il n'y a pas de réaffichage de la matrice qui ne soit contrôlé par le logiciel d'application.

Les primitives d'édition font apparaître une identification de la case de matrice et la variable à éditer. Pour cela, l'interface directe (côntrole d'E/S) avec le systeme est faite au moyen des primitives du langage PASCAL (read,write).

La mémoire nécessaire pour définir le menu et la matrice (définition des caractéristiques) est allouée dynamiquement. Les caractéristiques d'un outil de dialogue existant peut être modifié en mémoire sans affecter son affichage. Ainsi lors d'un changement de caractéristique, un encombrement écran relatif au dernier affichage doit être mémorisé pour permettre son effacement . Réciproquement, l'effacement sur l'écran ne correspond pas

obligatoirement à la destruction de ses caractéristiques, cela permet de "cacher" des entités.

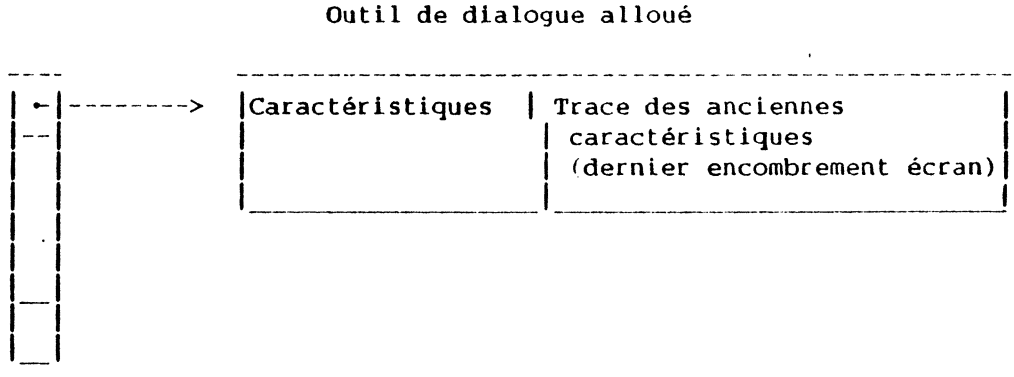


Tableau pointeurs allocation

La primitive Message :
Elle permet d'afficher une chaine de caractères à un position en coordonnées caractère suivant un attribut d'affichage (taille et inclinaison des caractères). Le positionnement dans les cases de matrice et le positionnement d'un message sont contrôlés à partir de la primitive de positionnement du curseur.

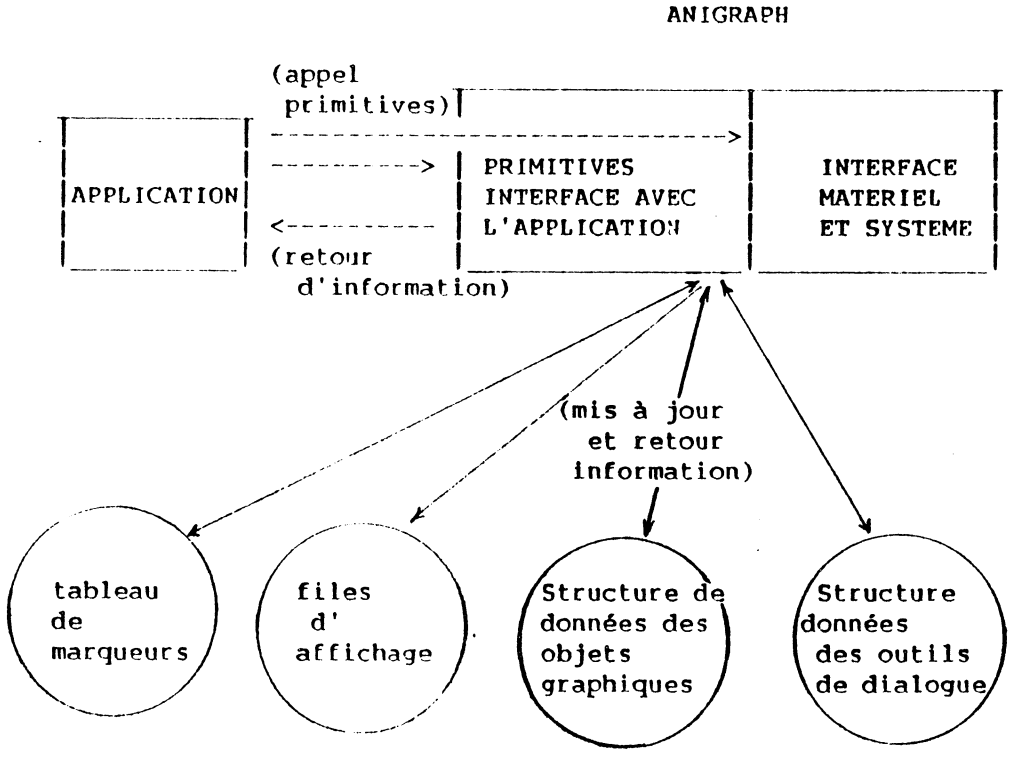
2.3.5. Structure du logiciel

Pour faciliter une portabilité et une prise en compte d'une extension du matériel graphique nous structurons le logiciel de base en deux parties :
- une interface avec l'application,
- une interface avec le matériel et le système d'exploitation.

L'interface avec l'application contient les primitives permettant de décrire les objets graphiques et les outils de dialogue. Néanmoins des primitives de l'interface avec le matériel sont accessibles à l'application pour contrôler directement une partie de la scène sans passer par ces objets ou ces outils.
Une fonction importante de l'interface avec le matériel est la génération de la file d'affichage en fonction du terminal utilisé.
L'interface avec le système d'exploitation est nécessaire pour cônteroler les E/S (ex. écriture de primitives simples en langage

machine pour la sortie rapide des caractères) et pour inhiber des fonctions relatives à l'utilisation de la console système comme terminal graphique (mode interruption ou non sur entrée de caractères, inhibition messages d'erreur ou non, ...).

Le lien des primitives avec l'application et la structure de données est schématisée sur la figure suivante :



2.3.6. Les primitives de ANIGRAPH

Nous décrivons ici l'ensemble des primitives graphiques nécessaire à l'interfaçage avec l'application.

Ces primitives regroupent :
- l'initialisation des données,
- la définition de clôture et fenêtre pour les objets graphiques,
- la création de marqueur et la gestion de leur affichage,
- la modification des attributs d'un objet-graphique ou d'un outil de dialogue,
- la destruction d'objet ou d'outil,

- l'affichage d' objet et d'outil,
- l'identification d'objet ou d'une fonction de menu ou d'une case de matrice,
- la lecture et l'édition de valeur dans une case de matrice.

1) Primitives relatives aux objets graphiques

```
iniobjets;
cloture (num_cloture, clotur);
fenetre (num_fenetre, fenetr);
marqueur (num_marqueur, marq);
assimarqueur (num_marqueur, obj, fig, sect);
listmarqueurs (obj);
modesection (obj, fig, sec, mode);
affsection (obj, fig, sect, num_cloture,numfenetre);
effasection (obj, fig, sect);
cachsection (obj, fig, sect);
rempbufsection (obj, fig, sect, buf_point_ou_texte, lg_buf,
descript, encombr);
identsection (obj, fig, sect);
```

2) Primitives relatives à l'outil Menu

```
inimenus;
rempmenu (num_menu, menu);
affmenu (num_menu, position);
effmenu (num_menu);
cachmenu (num_menu);
identfonction (num_menu, fonction);
```

3) Primitives relatives à l'outil Matrice

```
inimatrices;
rempmatrice (num_matrice, matrice);
affmatrice (num_matrice, position);
effmatrice (num_matrice);
cachmatrice (num_matrice);
identcase (num_matrice, caseidentifiee);
edmatentier (num_matrice, case, entier);
edmatreel (num_matrice, case, reel);
edmatchaine (num_matrice, case, chaine);
```

4) Primitives relatives aux messages

```
message (messag, descripteur, position_aff);
effecran (position, longzone, largzone);
```

Passage des paramètres d'entrée:

Les paramètres num_menu, num_matrice,obj, fig, sect prennent la valeur :

. -1 pour une entité non encore existante,
. >0 pour une entitédéjà existante.

Pour le triplet (obj, fig, sect) nous adoptons le protocole suivant :
a) sect = 0 <=> toutes les sections de la figure(obj, fig);
b) fig = 0 <=> toutes les figures de l'objet obj;
c) obj = 0 <=> tous les objets.

2.3.7. Les procédures de dessin

Ces procédures permettent la génération d'un ensemble de coordonnées de points correspondant au contenu d'une ou d'un ensemble de sections. Ces procédures servent à l'affichage d'un dessin au trait qui entre dans la représentation "figurative" (ex. celle d'un élément de l'objet de l'application) ou d'un symbole graphique qui entre dans la représentation "symbolique" (ex. celle d'un élément symbolisant une relation gestuelle entre l'utilisateur et l'objet dans le cas de ANIMA). La constitution d'une bibliotèque de ces procédures permet de faire des mises à jour indépendamment de l'application (suppression, adjonction, modification). Cette bibliothèque peut être commune à plusieurs applications. Nous pensons qu'il est utile de l'intégrer au niveau du logiciel de base. Les paramètres d'appel de ces procédures sont de type analytique : ex (rayon,centre) pour un cercle. Dans le cas d'utilisation actuelle dans ANIMA ils résultent d'une transformation simple (en général linéaire) des paramètres mécaniques et spatiaux associés aux objets.

2.3.8. Les outils d'acquisitions

2.3.8.1. Le capteur

La notion de Capteur introduite ici reprend l'idée de conception d'outils logiciels indépendants de l'application capables de supporter des acquisitions "complexes" sans que celles-ci ne soient gérées au niveau du logiciel d'application.

Ces outils sont relativement peu dévelopés malgré un besoin accru d'interaction nécessitant de telles acquisitions. L'étude sur les outils d'entrée a fait intervenir notamment le concept de "machine logique d'entrée" ou "machine virtuelle d'acquisition" qui délivre des valeurs logiques ne réflètant pas directement la morphologie des organes physiques en jeu, ni les instants réels d'activation des organes par l'opérateur.

Ces outils peuvent apparaitre au niveau d'un logiciel de base, ou d'une bibliothèque particulière ou peuvent être intégrés au langage de programmation. On peut citer à titre d'exemple:

. le langage d'acquisition défini dans GKS qui fait intervenir des classes de machines logiques correspondant à des fonctions d'acquisition graphiques (acquisition d'une position, choix sur un menu, acquisition d'un élément graphique ...). A chaque classe est associé un des modes de fonctionnement suivants :
- attente d'une action opérateur,
- lecture de la valeur logique courante,
- mode interruption sur action opérateur [28].

. les machines virtuelles simples ou composées développées par Ed Anson qui permettent de supporter des actions simples ou complexes pouvant faire concourir plusieurs dispositifs physiques. Il fait intervenir la machine d' acquisition composée qui met en jeu une interaction entre des machines simples. La communication se fait soit à partir de test et mise à jour de variables d'état (communes ou locales à chaque machine simple) soit à partir d'événements (ex. interruption) ou du traitement de ces événements. La définition de fonction de communication et des procédures de traitement d'évènements est prévue au niveau du compilateur [19].

. le modèle proposé par Jan Van den Bos qui permet de définir statiquement des expressions grammaticales de type hors-contexte à partir de machines logiques simples. Les opérateurs dans les expressions sont du type : boucle conditionnelle, sequencement, disjonction [51].

Sur ces différents outils le modèle Anson parait plus adapté à l'utilisateur dans le sens où il opère "naturellement" sur des objets qui sont les organes physiques (moyennant une bonne programmation des machines logiques par rapport aux besoins d'acquisition), mais non suivant des contraintes de langage que nécessitent les autres modèles.

2.3.8.2. Implantation : Le capteur-recherche

Le capteur-recherche est un outil spécifique inspiré du modèle de la machine d'acquisition composée définie par Ed Anson qui permet de gerer une boucle d'acquisition et d'acquérir une valeur finale suivant une action de l'opérateur sur les périphériques mis en jeu. Cette acquisition est relative à la recherche d'un paramètre à affecter à un objet dont une représentation sensorielle est perçue pendant la boucle d'acquisition. On s'interesse ici à la représentation graphique, les actions sont effectuées sur un capteur de force 3D (sujet au traitement itératif) et un clavier alphanumérique (utilisé pour le contrôle de l'itération).

Le capteur-recherche regroupe les éléments suivants:

. primitives d'initialisations
. primitive rendant la valeur logique
. liste de procédures de représentation. Ces procédures peuvent être propres au capteur ou apparaitre dans une bibliothèque qui interface le capteur au logiciel d'application pour la représentation graphique à adopter. Nous utiliserons ici une partie de l'ensemble des procédures de dessin,
. procédures de traitement d'évènements,
. variables d'état.

Les primitives d'initilisation permettent de définir la configuration des organes physiques mis en jeu (mode interruption ou non sur frappe d'une touche du clavier alphanumérique, fréquence d'horloge d'acquisition,...), le démarrage du processus d'acquisition (initialisation IT, demarrage convertisseur...), et l'initialisation des variables d'état.

Les primitives de traitement d'évènement répondent à des changements d'état dûs à une action externe (ex. action de l'opérateur) ou interne (ex. écoulement d'un quantum de temps, fin d'une conversion).

Les procédures de représentation permettent de faire une représentation "dynamique" de l'objet en fonction de valeurs courantes dans la boucle d'acquisition.

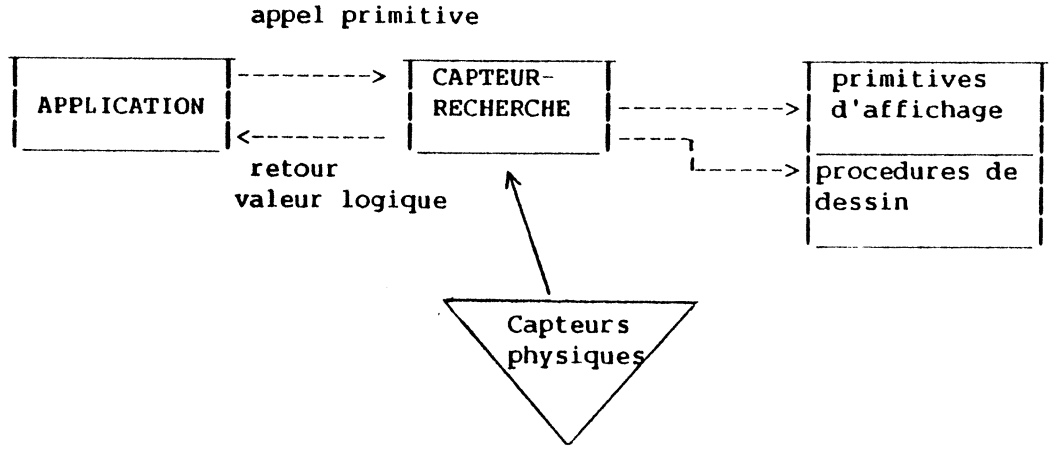
La primitive qui rend la valeur logique est caractérisé par deux parametres d'appel qui sont :
. identification de l'objet représenté
. une procedure de représentation.

Nous décrivons ici le schéma de principe du Capteur-recherche.

```
Schema Primitive_init;
configurer_clavier_en _IT
configurer_horloge_en_IT
demarrer_horloge
FinSchema


Schema Proc_IT_horloge
fairè_conversion(valeurs_converties);
FinSchema
```

```
Schema Proc_lire_courant(valcour);
valcour <- valeurs_converties;
FinSchema


Schema Proc_IT_clavier
arret <- caractere_frappe = caractere_arret_recherche;
FinSchema


Schema  primitive_capteur_rech(Val_capteur,  ident_objet,  representa-
tion)

  Tant que non arret faire
        effacer_representation(ident_objet);
        proc_lire_courant(val);
        representation(val,ident_objet);
        afficher(ident_objet);
        Val_capteur <- val;
  Fintantque
Finschema
```

Dans le contexte de l'application, nous pouvons illustrer le rôle du Capteur-recherche comme suit :



Remarque :

L'acquisition de coordonnees sur une tablette graphique fait

intervenir un capteur-recherche "cablé". La procédure de représentation fait intervenir généralement l'affichage d'un croix ou d'un réticule sur l'écran associé à la tablette. L'arret de la boucle d'acquisition correspond à la frappe d'une touche sur le clavier validant la "digitalisation".

2.3.9. Conclusion

Nous pensons que le logiciel de base réalisé correspond à un noyau qui est en adéquation avec les besoins de l'application actuelle.
Soit pour satisfaire à une évolution de l'application actuelle, soit pour s'adapter à une autre application. l'ensemble des primitives peut être étendu par exemple par :
. l'introduction de primitives d'affichage plus évoluées comme la "loupe" sur une portion d'écran définie de manière interactive qui facilitera les opérations de désignation,
. l'archivage des objets graphiques,...
Notons que le type des objets traités est simple ainsi que les fonctions adoptées mais cela a permis des traitements rapides et donc un degré d'interaction élevé entre l'entrée des commandes et leur exécution.

L'adjonction d'un attribut au niveau de la section, permettant d'afficher une surface pleine, peut être envisagée sur la version installée sur le VAX 730 qui fonctionne actuellement avec un attribut de couleur. Cet attribut permettra de faire l'association couleur-type d'élément représenté et faciliter le répérage d'éléments à désigner. Par ailleurs la possibilté d'avoir des surfaces pleines permettra de représenter des objets en pseudo 3D (objets appartenant à plusieurs plans parallèles) sans faire un calcul des surfaces cachées, mais en jouant sur la priorité d'affichage (par affectation d'un indice de priorité d'affichage sur des noeuds représentant les plans). Il va sans dire que la représentation de ce type d'objet va de pair avec une simulation mécanique et une visualisation dynamique soulevant les problèmes sur le traitement de plusieurs plans parallèles que nous ne verrons pas ici.

2.4. Outils pour l'interface utilisateur

2.4.1. Quelques concepts sur la définition de l'activité gestuelle

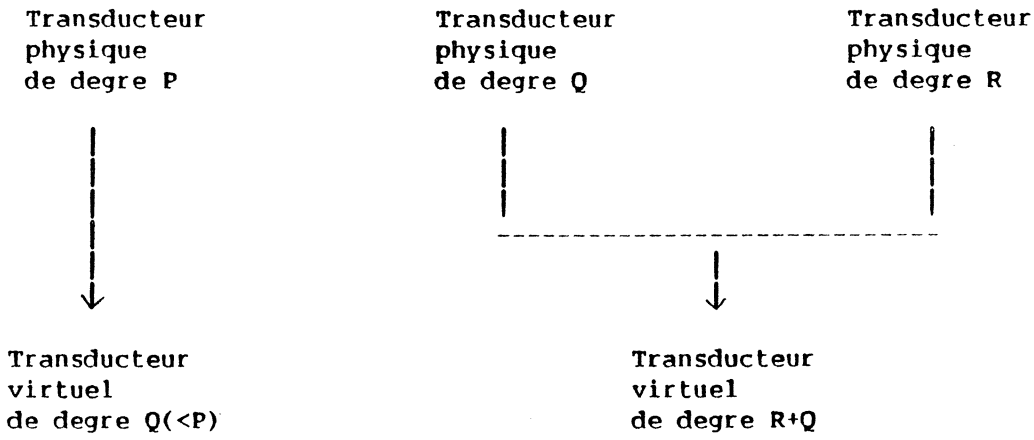
a) Transducteur virtuel et Fichier-geste

La réalisation de transducteurs gestuels est en progression en raison de l'importance des capteurs tactiles et du retour d'effort dans les domaines tels que la robotique et la conduite d'engins. Nous pensons néanmoins que d'importants problèmes mécaniques et technologiques subsistent, notamment dans la fabrication de transducteurs rétroactifs nécessitant pour le retour d'effort des moteurs puissants à faible inertie et à faible encombrement, et une morphologie permettant d'expérimenter un large espace gestuel.

Pour pouvoir faire abstraction de l'ensemble de transducteurs physiques adéquats et des contraintes techniques qu'impose leur réalisation, nous introduisons ici le concept de transducteur virtuel permettant de définir un transducteur "idéal" pour la simulation des fonctions gestuelles. Dans des conditions d'expérimentation où les transducteurs physiques sont totalement absents ces derniers peuvent être représentés par des fichiers de geste contenant des informations "synthétisées" du geste. Les caractéristiques des transducteur virtuels décrits et associés aux fichiers de geste permettent d'interpréter ces informations quant à leur type et leur organisation.

La notion de transducteur virtuel est une notion générale c'est à dire utilisable dans une simulation d'objet de dimension spatiale "quelconque". Le terme du transducteur adéquat traduit en effet l'adéquation entre le degré de liberté du transducteur et celui des objets simulés.

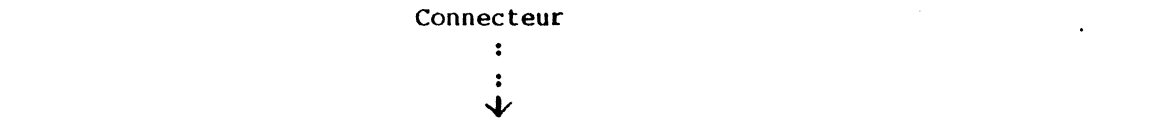
Nous schématisons ici quelques types de transformations possibles :

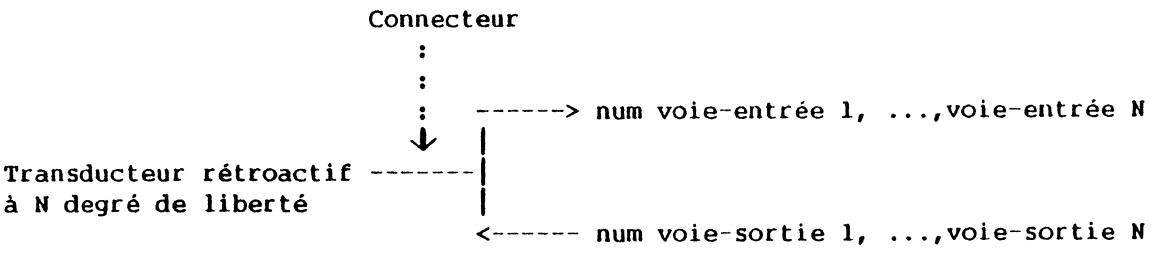


Ces remplacements sont d'ordre théorique, c'est à dire permettent le traitement de la simulation indépendamment des organes physiques connectés. En fait l'absence du transducteur physique adéquat pose quand même des problèmes d'adaptation si l'utilisateur veut faire une manipulation réaliste (ex. excitation d'un objet à deux degrés de liberté par deux transducteurs à degré de liberté unique).

b) Description du transducteur virtuel
Le transducteur virtuel est décrit à partir de numéros de voies physiques apparaissant sur la carte interface. Le transducteur non rétroactif à n degrés de liberté regroupe alors les numéros de voies en entrée. Un transducteur rétroactif comprend n numéros de voies physiques d'entrée et n numéros de voies physiques de sortie. La connexion des transducteurs physiques sur ces voies peut être fixe ou faite à l'initiative de l'utilisateur en tenant compte de la précédente description. Dans ce dernier cas, il dispose d'un choix sur un certain nombre de connecteurs enfichables reliant les E/S des transducteurs physiques et les E/S des convertisseurs.
Pour plus de simplicité nous adoptons pour le reste le terme de transducteur pour désigner le transducteur virtuel.

Nous illustrons ici la relation entre voies physiques et transducteur





c) Description des modules d'entrée

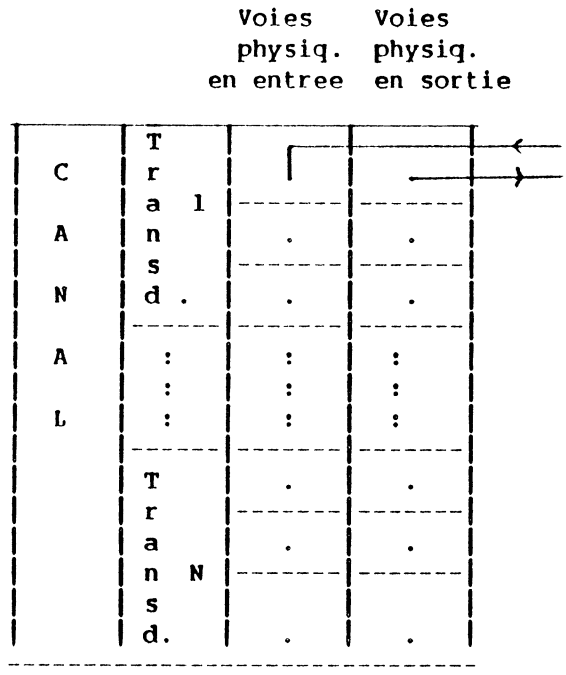
Elle se fait à partir de transducteurs existants. A un transducteur non rétroactif de degré n il peut être affecté n modules d'entrée (module PEM ou FEM) dont chacun prend en charge une voie de modulation. Un module d'entrée prenant en charge une voie d'excitation (module PEF ou FEP) peut être affecté à un transducteur rétroactif .

d) Description du canal et organisation du fichier-geste

d.1) Présentation

Nous avons vu qu' un canal regroupe un certain nombre de voies de modulation et d'excitation qui doivent observer les contraintes de complexité et de typologie gestuelle mentionnées au niveau du paragraphe(2.5.1 chap I). Il s'ensuit qu'un canal met en jeu au plus un transducteur retroactif et un certain nombre de transducteurs non rétroactifs. Le nombre total est limité par le nombre de voies phy-

siques disponibles au niveau du matériel.
Nous illustrons ici l'organisation en Canal - Transducteur - Voie physique.



Le fichier-geste est en quelque sorte l'image des transducteurs mis en jeu dans un canal. En effet chaque enregistrement du fichier-geste contient un ensemble ordonné des valeurs des voies physiques d'entrée (valeurs de conversion A/D) que regrouperaient les transducteurs physiques du canal.

Dans une phase d'expérimentation, chaque enregistrement du fichier correspond à l'ensemble de valeurs nécessaire à la phase de simulation mécanique.

d.2) Codage des informations dans le fichier-geste

Lorsque qu'un modification des signaux de gestes en provenance des capteurs n'intervient pas, on s'aperçoit que n enregistrements consécutifs peuvent être identiques. On adopte dans ce cas un facteur de multiplicité de n et un enregistrement unique pour remplacer la séquence. Un décompactage se fait lors de la lecture du fichier

sachant que l'ensemble des voies dans un enregistrement est utilisé n fois consécutifs par la phase de simulation mécanique. La structure de l'enregistrement du fichier-geste est alors le suivant :

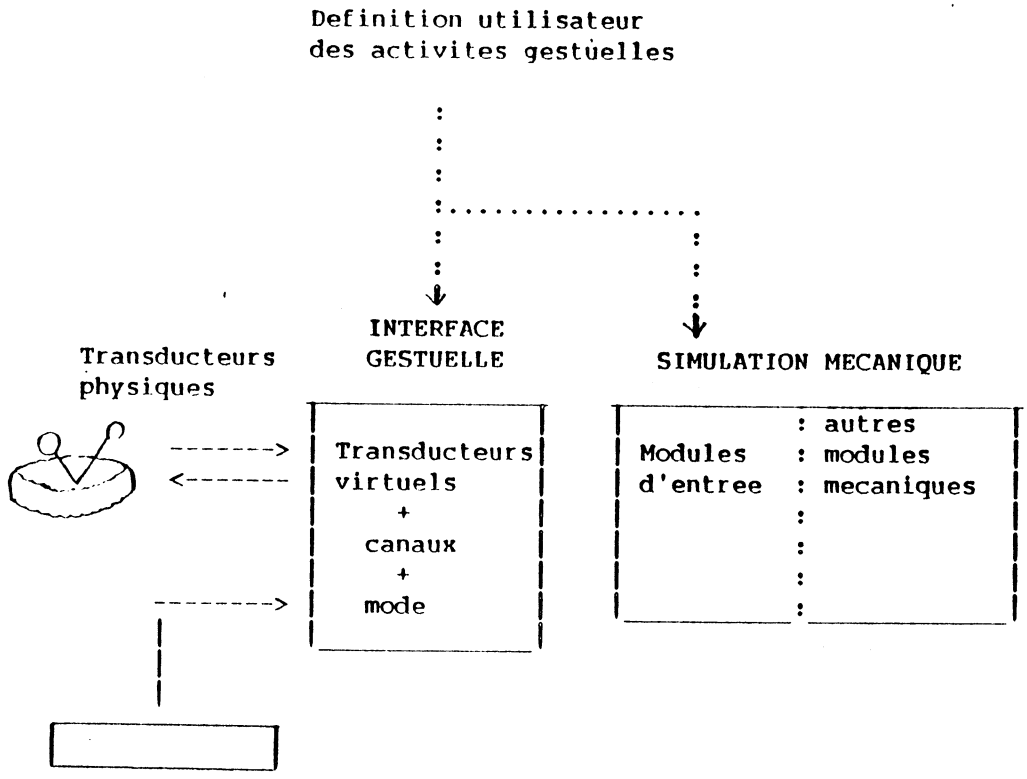
Ensemble ordonné de valeurs de conversion	Multiplicite

e) Modularité

Le langage de description du transducteur évoqué jusque là reste le même qu'il s'agisse d'utilisation effective de transducteurs physiques ou de fichier-geste. L'utilisateur doit néanmoins définir avant une phase d'expérimentation un mode des entrées gestuelles, relativement à ces deux cas , qui permettra de savoir s' il faut activer pendant cette phase un processus de conversion sur les voies physiques ou un processus de lecture sur le fichier-geste. Le schéma qui suit indique la répartion des informations rentrés par l'utilisateur dans l'environnement d'expérimentation. Une première partie des informations sert à spécifier l'interface gestuelle et contient la description :

- des transducteurs virtuels,
- des canaux,
- du mode des entrées gestuelles.

La seconde partie sert à spécifier le contexte du processus de simulation mécanique relatif aux modules d'entrée à mettre en jeu.



2.4.2. Entrée des paramètres et des arguments

2.4.2.1. Présentation

Nous distinguons essentiellement deux types d'information fournie par l'utilisateur. Il s'agit des arguments et des paramètres. L'ensemble de ces informations permet de mettre à jour la structure de données nécessaire à la représentation interne des objets. Celle-ci permet de :

. faire la représentation externe,
. générer l'ensemble des informations utiles aux procédures de simulation avant une phase d'expérimentation.
Nous distinguons par ailleurs deux types de paramètres qui sont les paramètres géométriques et les paramètres mécaniques. Les premiers sont relatifs au positionnement de l'objet dans l'espace, les seconds décrivent les valeurs des caractéristiques mécaniques (masse, raideur, frottement, pesanteur). Nous incluons dans ces derniers les coefficients de calibrage sur des entrées de force ou position qui adaptent les plages de variation externes des actions effectuées par l'opérateur avec les plages de variation internes au calcul de simulatiom mécanique.

La notion d'argument est introduite pour représenter une information de relation entre les entités suivantes :
transducteur, canal, voie physique convertisseur, module de visualisation, module matériel, module de liaison et module d'entrée.

Cette information de relation permet de définir l'aspect "qualitatif" des objets, contrairement à l'information sur les paramètres qui donne l'aspect "quantitatif".
L'affectation des arguments est définie de la manière suivante (liste des arguments pour chaque type d'entité source défini à gauche) :

. pour un canal, l'ensemble des transducteurs associés,

. pour un transducteur, son degré de liberte, son type (rétroactif ou non), l'ensemble des voies physiques de conversion concernées,

. pour un module de liaison, l'identification des éléments matériels qui constituent les attaches,

. pour un module d'entrée, le transducteur associé, l'identification de la voie physique dans le cas d'un module PEF ou FEM prenant en charge une voie de modulation, l'identification de l'élément matériel excité dans le cas d'un module FEP,

. pour un module de visualisation, l'identification des éléments matériels à représenter.

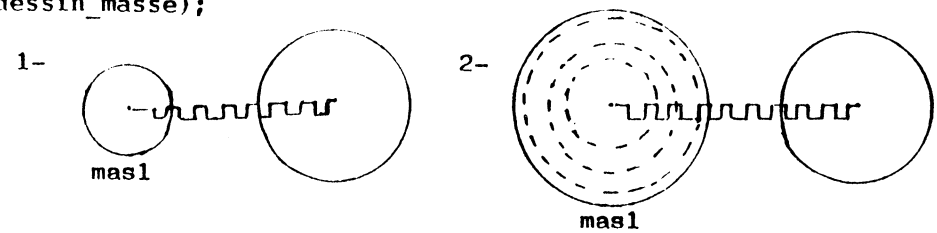
Ces relations sont directement mémorisées au niveau de la représentation interne. La structuration précédente des arguments constitue un choix, elle permet de faciliter le dialogue opérateur et d'éviter les redondances d'informations. Elle correspond à une optimisation de mémoire (par une réduction des données mémorisées) mais pas nécessairement à une réduction de traitement pour la représentation externe des objets ou pour la génération des informations nécessaires à la phase de simulation mécanique.

2.4.2.2. Entrée analogique de paramètres mécaniques

Cette entrée correspond essentiellement à l'utilisation du Capteur-recherche qui a été introduit dans le paragraphe (2.3.8.2 chap II). L'acquisition au moyen de capteur analogique permet :
- une souplesse d'utilisation,
- une affectation globale des paramètres des objets qui nous semble indispensable dans une première expérimentation du comportement simulé de ces derniers.

Voici un exemple d'appel au Capteur-recherche qui permet la mise à jour de la représentation d'un élément masse dont la section graphique est identifiée par (obj, fig, sect):

```
primitive_capteur_rech (valeur_masse_finale, obj, fig, sect,
proc_dessin_masse);
```



2.4.2.3. Entrée des arguments et de paramètres au moyen de chaines alphanumériques

Elle consiste notamment à faire une utilisation interactive de la Matrice qui a été définie au niveau du logiciel graphique de base pour le dialogue. Cet outil permet de faire facilement des choix multiples, des affectations, des remplacements de valeurs et une visualisation structurée d'information. Toute matrice affichée permet de faire une modification en boucle, à partir d'un pointage par le réticule jusqu'à acquittement par un caractère d'arrêt.
La séquence suivante écrite en pseudo-Pascal décrit le principe pour une interaction sur une matrice :

```
seq: créer_matrice (num_matrice,attribut_matrice);
     afficher_matrice (num_matrice);
     edition_contenu_matrice(num_matrice);
     tant que non acquittement faire
          identifier_case( case, caractere_frappe);
          acquittement<-caractère_frappe=caractere_d_acquittement;
          si non acquittement alors
             lire_dans_case(case, valeur);
             mis_jour_application(case,valeur);
     fintantque
fseq: effacer_matrice(num_matrice);
```

Les primitives de création, d'effacement, d'édition, d'identification et de lecture définies pour la matrice existent dans ANIGRAPH. Edition-contenu-matrice utilise les primitives d'édition.

Nous simulons l'existence d'une matrice de dimension importante pour laquelle légendes pouvent déborder de l'écran ou d'une clotûre adéquate pour l'affichage par un ensemble de pages, représentant les matrices effectives, que l'utilisateur peut faire tourner à l'aide des caractères "|" et "->". Ceci est important dans le cas où le nombre de paramètres à modifier est important.
Le schéma précédent est alors complété comme suit :

```
                .
                .
        si non acquittement alors
        cas caractere_frappe:
          "|","->" : effacer(num_matrice);
                si caractere_frappe="->"
                   alors generer_droite(attribut_matrice)
                   sinon generer_bas(attribut_matrice);
                   finsi
                creer_matrice(num_matrice);
                afficher_matrice(num_matrice,attribut_matrice);
                Edition_contenu_matrice(num_matrice);
        sinoncas
                .
                .
```

Remarques

Le schéma qui a été décrit est utilisé en grande partie pour entrer des valeurs numériques correspondant à des paramètres mécaniques. Il a été aussi utilisé pour définir :
- les valeurs numériques représentant les données de points de vue d'expérimentation (ex. vitesse de calcul, vitesse d'affichage).
- les arguments au niveau de la description des entrées gestuelles :
  . affectation de voies physiques aux transducteurs,
  . affectation canal - transducteurs,
  . affectation transducteurs - modules d'entrée.

2.4.2.4. Représentation alphanumérique de modules

Pour éviter une surcharge de la représentation figurative, nous avons été amenés à utiliser la matrice pour une représentation alphanumérique de liste de modules de simulation existants en mémoire. Le module est dans ce cas représenté par une ligne de matrice qui contient les arguments et les paramètres. La suppression se fait sur une boucle de pointage sur les lignes à partir du réticule. Nous ne

prévoyons pas de modification à partir de cette représentation. Dans le mode suppression, la matrice qui va contenir les éléments restants est affichée seulement à la fin des commandes de suppression. Les modules sont détruits avant ce dernier affichage.
Pour englober l'opération de suppression, nous complétons le schéma précédent comme suit :

```
            .
            .
        si non acquittement alors
           si mode_suppression alors
                si caractere_frappe=arret_suppression alors
                        detruire_elements( tab);
                        effacer_mat(num_matrice);
                        generer(attribut_matrice);
                        afficher_matrice(num_matrice);
                        remplissage_matrice;
                sinon
                        enregistrer_suppression(case, tab);
                finsi
           sinon
                        .
                        .
```

La séquence :
```
  lire_dans_case;
  mis_a_jour_application;
```
décrite dans le premier schéma a été supprimée, dans la mesure où on n'admet pas les modifications.
La procédure Enregistrer_suppression(case, tab) permet de mémoriser les commandes de suppression dans le tableau tab. La procédure Detruire-éléments fait la destruction effective des éléments à partir de ce tableau.

2.4.2.5. Entrée des paramètres géométriques et entrée graphique d'arguments

2.4.2.5.1. Mode de dialogue

Le mode de dialogue qui est défini par l'utilisateur permet à celui-ci de :

- faire un choix sur certains types d'opération de base à effectuer. Nous verrons dans les exemples qui suivent que cela concerne notamment la manière d'entrér les informations, à savoir la manière graphique ou la manière alphagraphique,
- agir sur le type de représentation externe consistant à faire un "filtrage" sur les informations à percevoir :
  . Un premier aspect concerne l'action sur la représentation "figurative" permettant de cacher l'ensemble des informations du type "quantitatif".
  . Un deuxième aspect concerne l'affichage ou non des marqueurs. Cela évite de surcharger inutilement la représentation externe notamment dans le cas où l'opérateur effectue une commande ne nécessitant pas l'identification des éléments représentés (ex. désignation par la réticule).
  Le marqueur apparait ici comme un mnémonique constitué d'un code du type d'élément marqué et d'un numéro d'ordre de création de l'élément marqué.

2.4.2.5.2. Entrée des paramètres et des arguments

Il s'agit notamment d'assurer au moyen de ces entrées les deux fonctions suivantes :

. définir l'emplacement des éléments matériels en donnant pour chacun d'eux leur coordonnée dans le plan. Suivant le mode de dialogue, l'entrée peut être :
  - soit analogique, par pointage par la souris ou le réticule sur un point d'écran,
  - soit numérique, par l'entrée de valeurs numériques.

. désigner les éléments matériels qui constituent les attaches des éléments de liaison. Suivant le mode de dialogue cette entrée peut être :
  - soit graphique, par pointage sur la représentation figurative,
  - soit nominative, par l'entrée d'une chaîne alphanumérique représentant le marqueur. Dans celui-ci le marqueur est nécessairement affiché. Ce dernier type de désignation peut se substituer au pointage graphique dans le cas d'une surcharge de l'écran, qui rendrait délicate la reconnaissance de section représentant l'élément attache.

2.4.2.5.3. Ensemble des commandes évoluées

Les fonctions définies précédemment constituent des fonctions de base pour le traitement des commandes plus évoluées dans la construction des objets.
L'entrée des coordonnées est utilisée pour les commandes suivantes :

- création d'un module élémentaire ou d'un module primaire : définition de positions initiales pour les éléments matériels et pour les modules d'entrée assimilés à des modules matériels.
Dans le cas de création d'un module primaire l'entrée des coordonnées est aidée par l' affichage d'un menu où figure l'ensemble des éléments à créer. Les coordonnées entrées à ce niveau définissent en partie les conditions initiales pour la phase de simulation.
- déplacement d'éléments matériels : définition de nouvelles coordonnées.

L'opération de désignation d'élément est utilisée dans les commandes suivantes :
- création de module : la définition des attaches pour les éléments de liaison,
- suppression de module : désignation d'élément constituant le module à supprimer,
- déplacement d'éléments matériels : désignation de l'élément à déplacer.

2.4.2.5.4. Traitement des modules primaires

L'utilisateur a accès aux éléments constituant un module primaire, ce qui lui permet d'exécuter des commandes identiques à celles qui portent sur les modules élémentaires (ex. désignation d'une masse de corde, déplacement de cette masse, ...).

Pour raison de simplicité, une destruction partielle ou une adjonction d'éléments n'est pas pour le moment traité (ex. ajout ou suppression d'une masse dans une ligne). Celle-ci nécessiterait une mise à jour "coûteuse" des données graphiques et des donnés de la représentation interne du module qui consisterait pour la suppression de la masse de corde à : faire la suppression automatique des 2 liaisons voisines et faire la création d'une liaison entre les 2 masses voisines.

La désignation graphique nécessaire pour la destruction d'un module primaire revient à la désignation d'un des éléments constitutifs.

2.4.3. Structuration des commandes utilisateurs.

2.4.3.1. Commandes hierarchisées

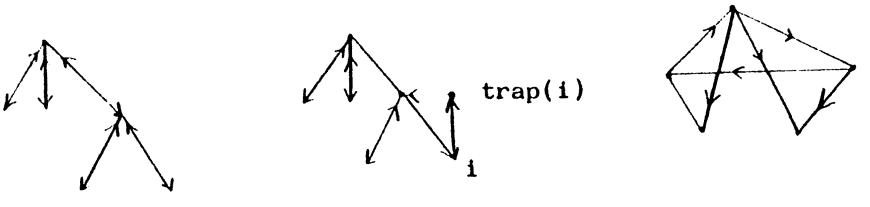
Les commandes sont organisées en classe et sous-classes pouvant être représentées par une arborescence des menus associés aux classes. Une commande non terminale permet d'afficher un menu fils à partir du menu courant sous une forme "en gigogne". Une commande terminale permet d'effectuer les actions suivantes :

- affichage d'une matrice en vue d'une modification sur la matrice, d'un parcours de la matrice ou d'une suppression de module,
- sortie du menu courant qui conduit à son effacement et au réaffichage du menu père (ce dernier est partiellement caché par le menu courant),
- exécution d'opérations d'entrée de coordonnées ou de désignation d'éléments tels qu'ils sont définis au paragraphe précédent.

La racine de l'arbre des menus représente une classe générale composée des sous-classes des commandes "clés" suivantes :
- mise à jour des caractéristiques de type qualitatif de l'objet,
- mise à jour des carctéristiques de type quantitatif de l'objet,
- mise à jour des points de vue de l'expérimentation,
- mise à jour du mode des accès gestuels,
- mise à jour du mode de représentation des résultats,
- démarrage d'une expérimentation temps-réel,
- arrêt de l'expérimentation temps réel en cours.

L'organisation hiérarchique des menus (retour au menu père) peut sembler contraignant, mais il correspond à une facilité et à une sureté pour l'utilisateur. Au niveau du programme cette hierarchie reflète tout simplement les appels et les retours de procédure de traitement des menus consécutifs. Une autre attitude consisterait à définir un certain nombre de classes de commandes indépendantes et avoir un automate d'états finis pour gérer les passages entre les classes. Un tel fonctionnement demande un effort à l'utilisateur pour avoir une perception globale des commandes disponibles à chaque instant. On se contentera dans notre cas, pour alléger le parcours des menus, d'introduire des menus "trappes", c'est à dire des menus sur lesquels on passe uniquement en remontant l'arbre des menus. Une application de ce principe a été par exemple l'affichage du menu comportant des commandes d'archivage en sortie du menu comportant les commandes de modification.

Nous faisons figurer ici les trois cas de parcours cités :
1- parcours arborescent
2- parcours arborescent avec menus trappes
3- parcours géré par un automate d'états finis



1- 2- 3-

Nous donnons ici le schéma de principe qui permet de traiter le parcours arborescent:

```
Schéma traiter(classe);

global menupere;
local arret, menucourant, commande, attribut_courant;
externes generer_attribut,traiter_terminal, terminal, sous_classe;

  arret <- faux;
  effacer_en_partie (menupère);
  generer_attribut (classe, attribut_menu);
  creer_menu (attribut_menu, menucourant);
  afficher_menu (menucourant);
  tant que non arret faire
      identifier (commande, menucourant);
      si terminal(commande) alors
              menupere <_ menucourant;
              traiter (sous_classe(commande));
      sinon   si commande = commande_sortie alors arrêt <- vrai
              sinon traiter_terminal (commande, classe);
      finsi
  fintantque
  effacer_menu (menucourant);
  afficher_menu (menupere);

finschéma
```

La fonction est appelée avec : traiter(classe_racine), après une initialisation de la variable menupere. Les primitives de création, affichage, effacement et identification sur les menus sont dans Anigraph.
L'existance de menus trappes conduit à compl'eter le schéma précédent

simplement comme suit :

```
        .
        .
traiter (sous_classe(commande));
si existe_menu_trappe(ss_classe, classe_trappe)
   alors traiter(classe_trappe);
   finsi
        .
        .
```

2.4.3.2. Automate de prépositionnement sur le menu

Des cheminements sur l'arbre des menus sont privilégiés pour faciliter les manipulations pour l'utilisateur. Ceci correspond à un prépositionnement du réticule sur le menu courant, à partir d'un automate qui le gère pour des chemins de longueur <= 2. Ces chemins peuvent être détectés "facilement" après une première expérimentation du système, en particulier ceux de longueur 2 au niveau du passage par la racine (classe générale des commandes). On remarque en effet à ce niveau quelques automatismes :

- passage de l'état de définition quantitatif à une expérimentation temps réel,
- passage de l'état de définition qualitatif à l'état de définition quantitatif,
- passage d'une expérimentation temps réel à l'état de définition quantitatif,...

Nous n'avons pas pour le moment prévu une programmation interactive de cet automate par l'utilisateur.

2.4.4. La structure expérimentable

Nous définissons dans cette section les données nécessaires à la représentation interne à un objet. Il s'agit notamment des données qui sont élaborées à partir des commandes de mise à jour des caractéristiques de type quantitatif et de type qualitatif. Les autres données (contexte d'expérimentation, modes), indépendantes de celles-ci et pouvant être communes à un ensemble d'objets, n'ont pas une structuration particulière (ensemble de variables réelles, entières et booléennes). Nous abordons leur description au niveau du chapitre sur les processus en phase d'expérimentation. L'indépendance entre ces 2 types de données permet de faire un archivage dans des fichiers indépendants.

a) Définition

Nous appelons structure expérimentable un ensemble d'informations contenu en mémoire centrale qui regroupe :

- une description des canaux d'entrées-sorties gestuelles,
- une description des transducteurs virtuels gestuels,
- une description des modules d'entrée,
- une description des modules mécaniques,
- une description des modules de visualisation.

Une description peut être vide. La structure expérimentable permet de faire une expérimentation cohérente c'est à dire qu'elle contient une syntaxe correcte sur la combinaison des éléments mis en jeu (ex: pas de liaison " en l'air"). De par la structure du langage, l'utilisateur dispose toujours d'une structure expérimentable au terme d'une commande. Aucune limitation n'est par contre imposée pour les valeurs de paramètre : on admet que des paramètres mécaniques qui sortent de la plage de valeurs dans laquelle le modèle est valide appartiennent à la structure. L'étude de cette plage qui relève des problèmes d'instablité du système mécanique (relatifs notamment à la discrétisation temporelle) n'est pas abordé dans ce travail.
La structure expérimentable est toujours accompagnée d'une représentation minimale sur l'écran. L'état de cette représentation est dépendante de la commande en cours.

b) Structure de données

La structure expérimentable est représentée sous forme de listes linéaires dont l'allocation dynamique est aisée par le langage Pascal, lequel permet de faire varier la taille des éléments de liste allouées. La structure contient les six têtes de liste suivantes :

- liste des canaux d'entrées-sorties gestuelles,
- liste des transducteurs gestuels,
- liste des modules d'entrée,
- liste des modules de liaison,
- liste des modules matériels,
- liste des modules de visualisation.

Cette répartition permet de :

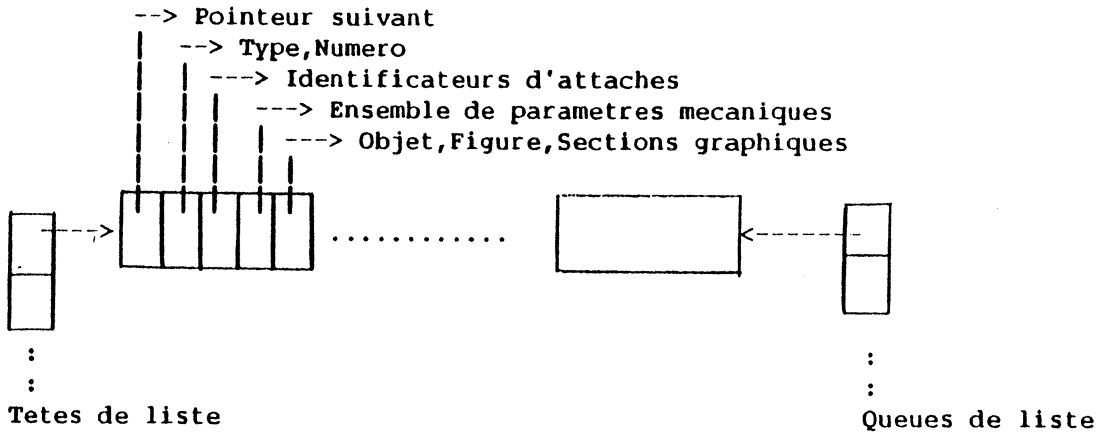
- faire des accès rapides pour les fonctions particulières à chaque liste (ex recherche des attaches dans la liste des modules matériels, représentation particulière des modules de visualisation et des modules d'entrée ...),
- organiser différemment chaque liste. Nous adoptons ici des listes ordonnées pour les modules matériels et les modules d'entrée, facilitant la mise à jour des numéros d'attaches (attaches représentés par des éléments matériels présents dans ces listes) dans le cas

d'une renumérotation de tous les modules.

L'ordre sur les listes nous amène à ajouter un pointeur de queue de liste facilitant les insertions en fin de liste. Le numéro de module cité précédemment permet d'identifier les cellules de liste et correspond aussi au numéro figurant sur le marqueur de la représentation graphique. Ce numéro fait partie du champ identification d'un élément de liste, lequel comprend :

- un identificateur,
- des arguments,
- des parametres mécaniques,
- une interface avec ANIGRAPH pour la représentation graphique.

Nous illustrons ici un élément de liste représentant un module de liaison.



c) Relation entre structure de données et simulation

La structuration des données de façon linéaire permet de traiter facilement l'isomorphie entre l'ensemble des modules de simulation mis en jeu et l'ensemble des appels aux procédures de simulation associées. Ce fonctionnement a été introduit dans les premières expérimentations qui ont conduit à définir le système actuel, lequel reflète une utilisation systématique du modèle linéaire dont on essayera d'en dégager les limites tant au niveau du langage de description des modules qu'au niveau des capacités de simulation.

Nous évoquons à titre indicatif et sans formalisation mathématique car cela dépasse le cadre de cette étude, un

fonctionnement possible où une "compilation" de la structure de données linéaire conduit à une génération rapide d'un algorithme solution du système d'équations manipulées actuellement. Cette solution doit permettre d'accroître la vitesse de simulation et être moins sujet à des de problèmes d'approximation que pose la discrétisation individuelle des équations en milieu continu.

Contrairement à cela, un autre fonctionnement consiste à assimiler les stuctures expérimentables à des "boîtes noires" que l'utilisateur peut interconnecter, "encapsuler" pour produire de nouvelles boîtes, sachant que les constituants de base d'une boîte sont les modules élémentaires ou primaires manipulés actuellement. La simulation dans ce cas peut :

- mettre en oeuvre les algorithmes simples actuels, consistant à "ouvrir" les boîtes et conserver le niveau dès modules définis actuellement,
- mettre en oeuvre un ensemble de solutions intermédiaires correspondant aux boîtes,
- mettre en oeuvre une solution générale qui intègre l'ensemble des boîtes.

Etant donné ces deux démarches, on peut distinguer deux orientations possibles, une qui consiste à faire une optimisation sur les mécanismes de la simulation, et une autre qui donne une richesse du langage de description des objets, et pour lesquelles il est judicieux de trouver un compromis.

d) Archivage

Les listes constituant la structure expérimentale peuvent être archivées dans un fichier structure. Celui-ci peut être reconstitué en mémoire centrale pour produire la même structure expérimentable. Une renumérotation des modules sera faite lors d'une restitution, cela à cause des "trous" relatifs à la numérotation dûs aux opérations de suppression.

e) Union de structures

Des opérations globales sur les structures paraissent nécessaires pour donner une souplesse à la fabrication de nouvelles structures. On mettra de côté ici les opérateurs qui permettraient de définir une "hiérarchie" ou une "intégration" sur les structures, car cette idée rejoint ce qui a été proposée au paragraphe ac). L'opérateur d'union qui a été implanté permet de garder une structuration linéaire des données. Cette opération, à partir de deux fichiers structure produit une structure expérimentable caractérisée d'une part par l'union des descriptions des modules mécaniques et des modules de visualisation et

d'autre part par une description vide sur les canaux, transducteurs et modules d'entrée. Cette opération d'union effectue :

- une génération en mémoire centrale des listes qui constituent les deux structures expérimentables,
- une fusion de listes (modules mécaniques, modules visualisation),
- une désallocation des éléments de liste décrivant canaux, transducteurs, modules d'entrée. La suppression de ces éléments permet d'éviter dans un premier temps de resoudre certaines incoherences au niveau des accès gestuels (non respect des contraintes d'un canal, conflit sur l'utilisation des voies physiques, ...). Les modules de visualisation et les modules de liaison ayant des arguments portant sur les modules d'entrée sont également supprimés.

Nous donnons ici le schéma de la procédure d'union de fichiers-structure.

```
Schéma union(structureunion);

lecture_memoire_listes(structure1);
lecture_mémoire_listes(structure2);
suppression_tranducteurs;
suppression_canaux;
suppression_modules_entrees;
suppression_liai_aux_entrees;
renumerotation(structure1, dernier1);
renumerotation(structure2, dernier2);

listedemodules_structure2 <- premier_struct2;
listedemodules_structure1<-premier_struct1;

Tant que non dernier_listedemodules_structure2 faire
     element2 <- premier_el2;
     tant que non dernier_element2 faire
          champ_ident<-champ_ident+dernier1[ordr(listedemodules_struct1)]
          Pour tout argument faire
               champ_ident_argument<-champ_ident_argument+
                                    dernier1[ordr(listedemodules_mat_struc(1)]
          Finpour
          element2 <- suivant_el2;
     fintantque

     listedemodules_structure1<-suivant_struct1;
     listedemodules_structure2 <- suivant_struct2;
fintantque

chainage_listes(structureunion);
generer_representation;
```

Finschéma

Remarques

Nous utilisons ici les primitives premier, suivant, et dernier permettant d'avancer et de faire un test de fin sur une file séquentielle de listes linéaires ou d'élément de liste. La fonction "ordr" donne pour une liste donnée un indice dans le tableau dernier1. Une indication sur cette fonction et sur les tableaux derniers1 et derniers2 est faite ci-dessus.

f) Renumérotation des modules

Il s'agit de la renumérotation de tous les modules dans une structure expérimentable. Elle est effectuée après la reconstitution des listes lors d'une opération de restitution de fichier-structure ou lors d'une opération d'union de stuctures, ceci évite de faire croître indéfiniment les numéros qui identifient les modules en supprimant les "trous" occasionnés par les opérations de suppression.

Nous décrivons ici le schéma de la procédure, avec le paramètre de retour "tabnewident" indiquant le tableau des derniers numéros affectés + 1, pour chaque liste renumérotée.

```
Schéma renumérotation(structure, tabnewident)

Pour tout listedemodules faire
     tabnewident[ordr(listedemodules)]<- 1;
     element <- premier;
     tant que non dernier_element faire
          champ_ident<-tabnewident[ordr(listedemodules)];
          si liste_contenant_attache(listedemodules) alors
             Pour tout listecontenantargument faire
                  element_arg <- premier_el_arg;
                  tant que non dernier_element_arg faire
                       Pour tout argument faire
                            si champ_ident_argument=sauvident alors
                              champ_ident_argument<-tabnewident[ordr(listed
                            finsi
                       Finpour
                       element_arg <- suivant_el_arg;
                  Fintantque
          finsi
          tabnewident[ordr(listedemodules)]<-tabnewident[ordr(listedemoduel
          element<-suivant_element;
      fintantque
Finpour

Finschéma
```

Remarque
-------- La fonction "ordr" indique l'ordre de visite de chaque liste pour sa renumérotation.

g) Affectation globale de paramètres

Elle s'effectue à deux niveaux : - par une affectation par défaut des paramètres à chaque création d'un module. Les paramètres appartiennent dans ce cas à une plage valide pour la simulation mécanique,
-par des commandes de l'utilisateur qui lui permettent les possibilités suivantes :
. affectation d'un ensemble de paramètres pour tous les modules d'un même type,
. affectation d'un ensemble de valeurs pour les modules d'un même type ayant un ordre de création situé dans une intervalle donnée. Les bornes de celle-ci correspond à des numeros identificateurs apparaissant sur le marqueurs.

g) Classe de commande INIT

La possibilité pour l'utilisateur d'"initialiser" une structure expérimentable à partir de fichiers nous conduit à introduire une classe de commande particulière au dessus de la classe-racine qui a été définie dans l'arborescence des commandes. Les commandes de cette classe permettent d'effectuer une des trois opérations suivantes :

. restitution d'un fichier structure,
. union de structures,
. désignation d'une structure vide,

et passer automatiquement sur le menu de la classe-racine.

Par ailleurs la sortie de la classe-racine implique successivement :

- un archivage automatique de la structure courante,
- un effacement de sa représentation à l'écran,
- une désallocation de la mémoire attribuée à celle-ci,
- le retour à la classe INIT.

La sortie de la classe INIT permet de revenir sur la classe de commandes du système d'exploitation.

Compte tenu de la classe INIT, nous décrivons ici le schéma général du programme de dialogue.

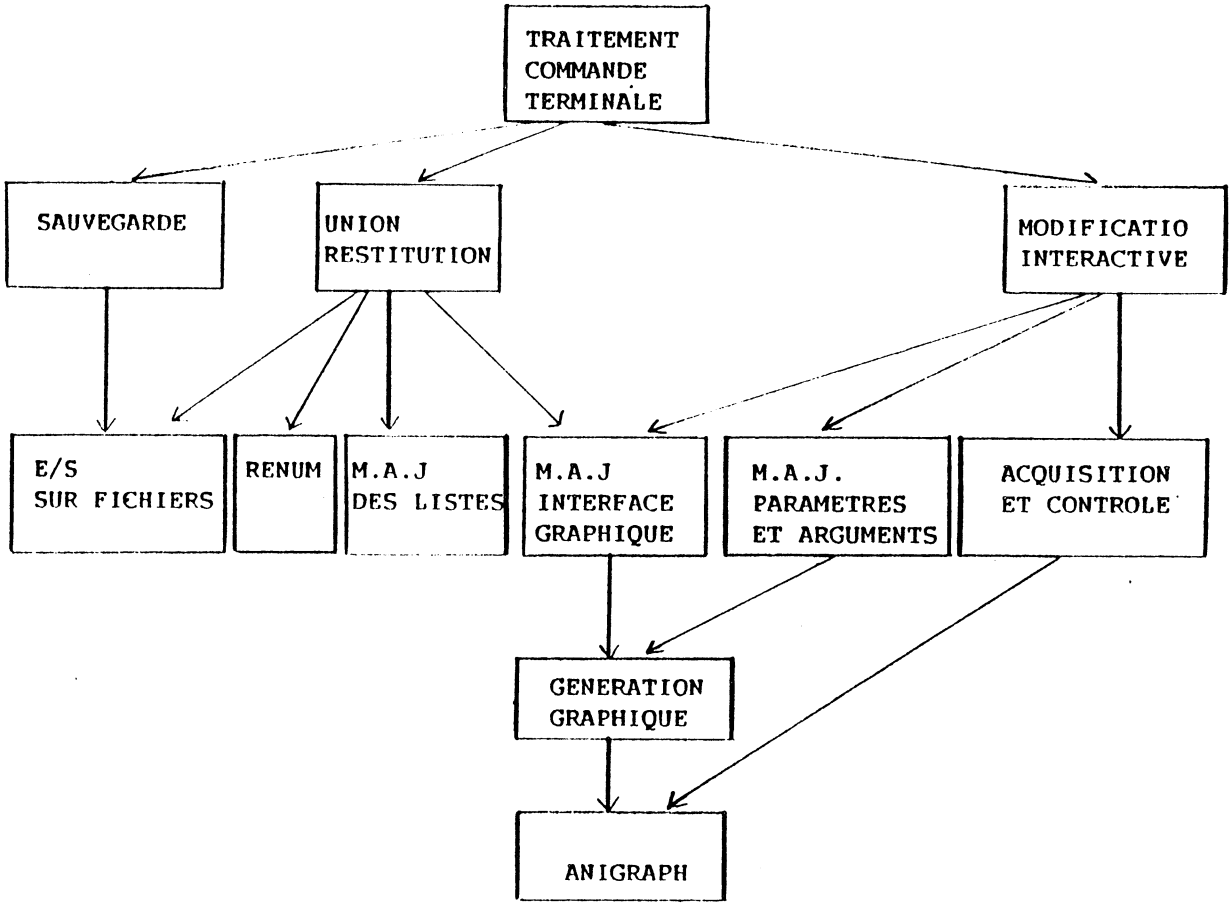
```
Schéma Anima

global menupere;

generer_attribut (menuinit);
creer_menu(menuinit);
sortie_init<-faux;

tant que non sortie_init faire

     afficher_menu(menuinit);
     identifier(fonction,menuinit);
        cas fonction dans
            lecture : lecture_structure;
            union   : union_structures;
            structure_vide : ;
            sortie_menu    :sortie_init<-vrai
        fincas
        si non_sortie_init alors
           menupere<-menuinit;
           traiter(classe_racine);
           archivage_automatique;
           desallocation_memoire_structure_courante;
        finsi

fintantque
effacer_menu(menuinit);

finschéma
```

2.4.5. Architecture du programme de dialogue

Nous adoptons une architecture modulaire permettant de dissocier essentiellement les fonctions suivantes :
- mise à jour des données (création et destruction d'éléments de liste, mise à jour des paramètres et arguments),
- mise à jour de l'interface avec le logiciel graphique et gestion de la représentation graphique.

Cette dissociation est profitable pour faire le traitement des deux types de commande suivants :
- création d'une structure expérimentable à partir d'une lecture sur fichier, ou d'opération sur fichiers
- modification d'une structure expérimentable existante.

Nous décrivons ici l'ensemble de blocs fonctionnels qui ont permis de réaliser cette modularité.



2.4.6 Description de procédures

Nous décrivons ici quelques procédures types utilisées dans le programme de construction des objets.

I- Procédure de recherche d'une attache de liaison

Le schéma suivant décrit une procédure faisant l'acquisition des

arguments. Il s'agit de la recherche d'une attache pour un élément de liaison ou un élément assimilé à une liaison.
Le paramètre "ident_element" est indispensable pour identifier l'élément matériel constituant l'attache dans un module primaire (ex. l'identificateur d'une masse de corde).
Pointeur indique le pointeur sur l'élément de liste représentant le module.

```
Schéma recherche_attache (pointeur,ident_element);

recherche_sectiongraphique(obj,fig,sect);
recherche (liste_materiels,obj,fig,sect,trouve,pointeur,ident_element);
si non trouve alors
   recherche(liste_entrees,obj,fig,sect,trouve,pointeur,ident_element);
finsi

Finschéma

Schéma recherche_sectiongraphique (obj,fig,sect);

trouve<-faux;
tant que non trouve faire

  message_operateur( signal_operation_pointage);
  identification_section(obj,fig,sect);

fintantque

Finschéma
```

Remarque

La recherche se fait dans les listes "liste_materiels" et "liste_entrees" qui peuvent contenir les modules matériels ou assimilés matériels. La procédure est utilisée dans un mode d'entrée graphique. Elle est évidente dans le mode d'entrée alphagraphique (acquisition de la chaine alphanumérique correspondant à l'identificateur).

II- Procédures de création

II-a Création de module élémentaire

Nous décrivons ici un schéma de procédure qui traite la création d'un module masse élémentaire. Cette procédure gère :
- l'acquisition d'arguments et de paramètres géométriques,
- l'allocation mémoire pour le module et leur mise à jour à partir de la première opération,

- la génération des données graphiques (au niveau Anigraph) et de l'interface graphique (au niveau de la structure expérimentable),
- la représentation externe du module.

Les fonctions ci-dessus sont communes à la création d'un module de type quelconque.

```
Schéma creation_masse;

si non mode_lecture-structure
   alors acquerir_coordonnees(point);
        allocation_element_liste(pointeur);
        mis_a_jour(pointeur, point,valeur_masse_par_defaut);
        insertion_en-queue(pointeur,queue_de_liste_materiels);
finsi
dessin_masse(point,valeur_masse,attribut_section);
mis_a_jour_donnees-graphiques(attribut_section, interface_graphique);
mis_a_jour(pointeur, interface_graphique);
representation_graphique(interface_graphique);

Finschéma
```

Modularité

La procédure décrite assure une partie du traitement dans le cas d'une restitution de fichier-structure relatif aux deux dernières fonctions citées qui ne sont pas prise en charge par la procédure de restitution lors de la reconstitution des éléments de liste en mémoire centrale.

II-b Création de module primaire

Pour la création d'un module primaire, les fonctions de la procédure de création sont exécutées en deux étapes :

- description d'un ensemble minimum d'éléments à partir de l'affichage d'un menu-fils,
- mise à jour automatique des informations et de la représentation graphique pour les éléments restants.

La mise à jour automatique s'applique notamment pour les éléments de liaison à partir d'une description géométrique des éléments matériels représentant les attaches. Chaque étape fait intervenir une boucle que nous décrivons ici dans l'exemple d'une création de corde :

- une boucle permettant la description des éléments matériels de la corde dans un ordre quelconque,
- une boucle permettant la mise à jour des informations sur les liaisons suivant un ordre fixé à partir d'une des extrémités de la corde.

Cette description nous amène au schéma suivant :

```
Schéma creer_corde

message ("lecture nombre de masses de la corde", nbmas);
generer_menu_corde(nbmas,menucorde);
fig_graphique<- -1;

Pour i=1 a nbmas+2 faire
     identifier(identelement,menucorde);
     acquerir(point_materiel);
     mis_a_jour(pointeur, point_materiel, valeur_masse_defaut,identelement)
     dessin_masse_ou_sol(identelement, attribut_de_section);
     sect_graphique<- -1;
     mis_a_jour_donnees_graphiques(attribut_de_section, interface_graphique
     representation_graphique(interface_graphique);
Finpour

Pour i=1 a nbmas+2 faire
     sect_graphique<- -1;
     mis_a_jour(pointeur,param_liaison_par_defaut,i);
     dessin_liaison(i,i+1,attribut_de_section);
     mis_a_jour_donnees_graphiques();
     representation_graphique();
Finpour

Finschéma
```

II-c Interface avec Anigraph

Une sous-fonction de la génération de l'interface graphique concerne la création des sections graphiques. Le schéma précédent montre l'initialisation d'une nouvelle figure (par fig-graphique <- -1) à chaque création d'un module primaire. Cela rend disponible toutes les sections graphiques de la figure qui sont associés à la représentation des éléments qui constituent le module. Dans le cas d'un module élémentaire cette recherche de section graphique se traduit par la recherche d'une section libre dans une figure ne représentant pas un module primaire.
On utilise à cet effet la variable globale fig-graphique-elem indiquant la figure couramment utilisée. Cette variable est mise à -1 à l'initialisation.

Le schéma de création de la section est alors :

```
Schéma creation_section_graph (attribut_sect,  section);

section<- -1;
Tant que fig_graphique_elem = -1 faire
     creer_section (obj,fi_graphique, section, attribut_sect);
     si section_graphique=-1
        alors fig_graphique <- -1
     finsi
Fintantque

finschéma
```

DEFINITION DE L'ENVIRONNEMENT D'EXPERIMENTATION

1. Introduction

Ce chapitre traite de l'étude de l'ensemble des processus à mettre en oeuvre dans une phase d'expérimentation de structure expérimentable. Nous reprenons ici le terme de processus sous son aspect "générique" mais nous n'attribuerons pas une description formelle mathématique des processus introduits, cela n'étant pas le but recherché ici. Nous avons utilisé une terminologie émanant des ouvrages de base existants qui décrivent le concept de processus dans un système informatique [26]. En effet, nous avons fait dans une première étape théorique, une approche d'une fonction à réaliser en terme de processus et ainsi nous avons essayé de définir pour l'ensemble des processus des conditions de parallélisme, de communication et de synchronisation.

Nous évoquons alors dans ce chapître :
- un certain nombre de fonctions pouvant être accomplies pendant la phase d'expérimentation d'un objet,
- des manières possibles d' articuler ces fonctions.

De l'articulation entre ces fonctions dépend la vitesse globale d'exécution, c'est à dire du caractère temps réel ou non temps-réel de l'expérimentation. Avant d'envisager une implantation réelle nous avons tenté de dissocier ces deux formes d'expérimentation qui d'une part correspondent à des situations spécifiques pour l'utilisateur, d'autre part ont nécessité des outils différents pour cette implantation.

2. Présentation générale des processus.

Nous introduisons les processus suivants qui relèvent à priori d'une certaine autonomie. Nous leur associons des noms de fonction que nous avons eu l'occasion de mentionner au niveau de la présentation théorique de notre système :

- processus "dialogue",
- processus "acquisition des échantillons de geste",
- processus "restitution du retour d'effort",
- processus "simulation mécanique",
- processus "visualisation",
- processus "mémorisation de l'expérience instrumentale".

Cette décomposition en processus nous conduit à développer les problèmes existants en terme de :
- processeurs physiques existants,
- systèmes de gestion de processus, existants sur ces processeurs,
- vitesse attribuées aux processus,

. vitesse des organes d'entrée-sortie mis en oeuvre par les processus.

2.1. Contraintes du temps réel

Les contraintes d'une expérimentation temps réel impose des vitesses aux processus et des instants précis de synchronisation. Cela peut ne nécessiter par contre qu'un minimum de ressources de synchronisation telle une horloge de référence par rapport à une gestion de processus complètement asynchrones ne faisant pas intervenir des mesures de durée mais une relation d'ordre sur les instructions (utilisation de sémaphores, sémaphores privés, boites aux lettres ...).

Par ailleurs nous ne discuterons pas ici de l'utilisation de modèles "complexes" tel que Grafcet[35], mettant en jeu des contraintes fortes sur les vitesses des processus coopérants et capable de réagir rapidement à des défectuosités de processus ou à des évenements extérieurs. Un tel modèle est utilisé dans des applications comme la conduite de rames de métro ou de processus industriels.

2.1.1. La boucle Geste-Vue

2.1.1.1. Présentation

Au niveau du système Anima l'objectif de base à atteindre est la satisfaction des contraintes de la boucle temps réel Geste-Vue. Nous nous fixons à ce propos une vitesse minimale d'acquisition des échantillons de geste à 100 HZ , d'une part pour répondre à une bande passante du dispositif à retour d'effort utilisé que nous prenons comme référence (des phénomènes d'oscillation apparaissent pour des vitesses inférieures) et d'autre part pour permettre l'utilisation de ces échantillons par le processus de simulation mécanique fonctionnant à vitesse élevée.
Par ailleurs, une vitesse maximale de visualisation est à expérimenter, d'une part dans un souci de qualité du mouvement, et d'autre part pour étudier les effets perceptifs d'une visualisation à grande vitesse, laquelle jusque là est généralement limitée à 25 HZ. Cette fréquence correspond en effet à une moyenne par rapport à la persistance rétinienne permettant la perception d'un enchaînement sur des images comme les images cinématographiques et a été adoptée pour réduire des contraintes matérielles (vitesses de conversion analogiques, vitesses du processeur graphique) et économiques (standart des support de film à 25 images/s).

Nous rappelons ici que les modules de visualisation permettent de générer l'image au moyen d'une transformation simple des variables positions appartenant aux modules de mécaniques. Nous appelons plus

spécifiquement échantillon visuel le résultat de cette transformation qui est constitué d' une succession de commandes élémentaires directement assimilable par le matériel de visualisation. La gestion du transfert de l'échantillon visuel est assuré par le processus de visualisation.

2.1.1.2. Le pas de simulation et la différenciation des fréquences

Le pas de simulation consiste à la résolution, à un instant discrétisé donné, de l'ensemble des équations qui régissent chaque module d'entrée, module matériel, module de liaison, module contrôle de paramètres mis en jeu.
Nous disons que l'échantillon visuel peut résulter d'un ou plusieurs pas de simulation suivant que les modules de visualisation s'exécutent après un ou plusieurs pas de simulation. Ceci détermine la fréquence de visualisation. Le nombre de pas de simulation exécutés pour un échantillon de geste acquis détermine la fréquence de simulation.
Nous disons alors que ces fréquences peuvent être variables et que :
- la fréquence de simulation peut être supérieure ou égale à la fréquence d'acquisition,
- la fréquence de visualisation peut être égale ou inférieure à la fréquence de simulation.

Nous verrons par la suite :
- le cas de fréquences non variables pour une expérimentation donnée, pour garder en partie une cohérence entre les états du système mécanique,
- des combinaisons particulières de ces fréquences, pour obtenir une "régularite" sur les images affichées.

2.1.1.3. Les processus de base

Il y a quatre processus cycliques à mettre en oeuvre pour une expérimentation de base : les processus d'acquisition des gestes, le processus de simulation, le processus de visualisation et le processus de retour d'effort.
Il est important pour satisfaire les contraintes du temps réel de :
- choisir des processeurs rapides pour l'exécution de ces processus,
- disposer d'une certaine rapidité au niveau des organes pilotés par les processus
- paralléliser au maximum l'exécution des processus.

Un mécanisme de synchronisation simple qui peut être introduit ici est une synchronisation sur chaque cycle d'une horloge commune pour laquelle il peut y avoir 0 ou n éxécutions de chaque processus. Un temps de cycle programmable de cette horloge pourra servir dans une phase d'évaluation à mesurer les débordements sur les temps d'exécution.
Le parallélisme sur l'exécution des processus devra respecter le

protocole de communication d'information suivant :
- accès aux échantillons de geste en exclusion mutuelle,
- accès aux echantillons de retour d'effort en exclusion mutuelle,
- accès aux échantillons visuels en exclusion mutuelle.

2.1.2. Extansion de la boucle Geste-Vue

2.1.2.1. Présentation

L'idée ici est d'introduire des fonctions qui nous paraissent essentielles pour étendre le champ d'investigation dans la situation de création nous permettant d'aborder une phase "compositionnelle"mettant en oeuvre des objets "complexes" et nécessitant des fonctions évoluées de mémorisation.
Nous caractérisons cette phase par les facteurs suivants :
- possibilité d'avoir une multiplicité importante des modules de simulation,
- possibilité de définir de modules de visualisation mettant en oeuvre des transformations complexes des variables position et dont la sortie des échantillons visuels nécessite un procédé de visualisation fonctionnant à vitesse lente,
- possibilité de mémoriser des échantillons visuels,
- possibilité d'utiliser une partie du code des processus de base.
Le travail actuel n'aborde pas l'étude détaillée des contextes de processus dans la phase "compositionnelle". Il s'agit notamment :
- de dégager les situations propre à cette phase,
- dégager certaines ressources nécessaire à cette phase qui doivent être constituées par l'exécution des processus de base,
- dégager les mécanismes de synchronisation nécessaires vu les vitesses des processus et des organes mis en oeuvre qui peuvent compromettre les mécanismes introduits dans un fonctionnement en temps réel.

2.1.2.2. Visualisation lente

La visualisation lente correspond à un affichage d'images complexes à vitesse basse, mais qui pourra profiter d'un traitement plus conséquent de l'image (ex. ajout de forme, habillage couleur, texture) comparativement à la visualisation temps réel. En effet cette dernière visualisation revêt surtout le caractère expérimental, la sortie des dessins au trait permet d'étudier plus facilement le comportement dynamique des structures simulées, et de faire un premier apprentissage pour l'animation. Nous supposons néanmoins que le traitement de la sortie visuelle complexe nécessiterait la participation d'un processeur spécialisé (processeur graphique) pour la puissance de calcul demandée. La vitesse de sortie des échantillons peut dans ce cas être inférieure à un seuil permettant d'apprécier une animation réelle et rendant difficile le contrôle gestuel, nous supposons

dans ce cas que :
- les échantillons visuels sont mémorisables sous forme de film ou de fichiers d'échantillons.
- les informations de geste proviennent de fichiers-geste.

2.1.2.3. Mémorisation d'échantillons visuels

Une mémorisation d'échantillons visuels est souhaitable dans un but de production et de composition d'images dans un mode "off line". Ces fonctions n'entrent pas dans la classe des fonctions abordées jusqu'ici. La production ou composition d'images devront profiter des caractéristiques du mouvement contenus au niveau de la mémorisation des échantillons visuels qui regroupe alors les échantillons de base. Ces fonctions peuvent être de simples restitutions.
La mémorisation peut être faite sur fichiers ou support audio-visuel ou support papier (table traçante, imprimante graphique,...). Les échantillons mémorisés sur fichiers peuvent être de nature différents :
a) échantillons directement restituables:
  1- mémorisattion d'images pleines nécessitant un codage pour un compactage des échantillons (ex. mémorisation des pixels et codage par plage de couleur pour des images couleur).
  2- mémorisation de commandes élémentaires pour des dessins au trait.
b) échantillons contenant des informations plus "globales" destinés à des traitements ultérieurs (ex. mémorisation d'une partie de l'ensemble des variables des modules mécaniques).

2.1.2.4. Utilisation de fichiers-geste

La constitution d'un fichier geste nécessite l'enregistrement temps réel sur mémoire de masse des échantillons de geste des échantillons acquis à partir des transducteurs physiques, lors d'une expérimentation en cours.
Le fichier geste résultant servira pour la production de tout ou une partie des échantillons de geste, soit dans une phase d'expérimentation temps réel, soit dans une phase "compositionnelle".
Cette mémorisation "objective" des gestes permettra de faire à terme :
- une analyse à partir des échantillons lus sur les transducteurs en vue d'extraire des caractéristiques dynamiques du geste,
- générer en conséquence des fichiers-geste de synthèse.
Des travaux dans ce sens sont actuellement amorcés au sein de l'équipe et doivent aboutir sur la définition d'un "éditeur" de fichiers-geste.

2.2. Synthèse des scénarios possibles

Nous avons évoqué jusque là un certain nombre de fonctions pouvant rentrer en jeu dans un environnement d'expérimentation. Il est important pour nous dans un premier temps d' analyser les mécanismes de

base qui peuvent relier les différents processus et permettra de déduire par la suite des articulations optimales.
L'étude de deux classes de fonctionnement possible nous paraît ici indispensables :
- une classe de fonctionnement "synchrone",
- une classe de fonctionnement "asynchrone".
La première classe mettra en oeuvre une articulation entre processus pour lesquels on a une estimation précise des vitesses et qu'on synchronisera sur une horloge commune. Nous incluons dans cette classe toute articulation permettant un affichage dynamique synchronisé sur une acquisition gestuelle et pouvant inclure des fonctions d'enregistrement rapides ("topage" d'une caméra, contrôle direct en DMA des E/S sur disques,...). La fréquence de visualisation peut dans ce cas varier de 0 à 200 Hz. Cette fréquence correspond à l'écart entre l'affichage de deux échantillons différents, la fréquence supérieure mentionnée ici est donnée à titre indicatif (temps d'affichage d'images très simples sur un écran à balayage cavalier) mais ne correspond pas à une borne à respecter.

La deuxieme classe de fonctionnement met en oeuvre des processus à vitesse moins élevée qui ne permettent pas d'atteindre une simultanéité apparente entre l'affichage des images et le geste introduit par l'utilisateur.
Les protocoles de communication de données introduites dans le paragraphe (2.2.2) sont toutefois à respecter, mais la synchronisation sur une horloge commune n'est plus adéquate en raison de la lenteur sur l'exécution de certains processus. Nous supposons que dans cette classe de fonctionnement :
- l'affichage d'échantillons visuels ne sert pas d'aide à l'apprentissage de l'animation mais de trace à l'exécution des processus,
- la restitution des informations de retour d'effort n'est pas indispensable,
- la production d'échantillons gestuels ne se fait pas à partir d'organes physiques mais se fait d'une façon "atemporelle", notamment à partir de lecture de fichiers-geste.
Le fonctionnement peut être ici approché par le modèle de Producteurs-Consommateurs] avec une longueur de message > 1 (échantillon) et faisant concourir des processus pour :
- la production et consommation d'échantillons gestuels,
- la production et consommation d'échantillons visuels.

2.3. Processus dialogue

Ce processus traite la communication interactive avec l'opérateur pour la construction ou la mise à jour des structures expérimentables. L'étude du code et des données manipulés par ce processus a fait l'objet des chapîtres précédents. Il est important de pouvoir

paralléliser l'exécution de ce processus avec ceux qui interviennent pour le "jeu" (expérimentation temps-réel), notamment dans une phase d'apprentissage où une boucle Modification-Expérimentation est souhaitée. Cette phase peut alors nécessiter le concours simultané de plusieurs opérateurs suivant la multiplicité des transducteurs gestuels disponibles.

L'interaction entre le processus dialogue et les processus "jeu" paraît à priori nulle (sauf dans le cas de concours à la ressource processeur) car ils traitent deux activités autonomes. Le processus dialogue aura néanmoins les privilèges de blocage ou déblocage des processus "jeu" suivant une demande d'opérateur par les commandes : arrêt jeu, démarrage jeu. Cette communication entre les deux groupes de processus peut être étendue à une communication de variables pendant la phase de jeu permettant à l'opérateur d'influer directement sur le comportement des processus "jeu" par une modification de leur contexte. Cette action aboutit notamment à la notion de "modification de structure en temps-réel" que nous abordons plus loin.

Pour l'utilisateur le parallélisme entre l'exécution du processus dialogue et des processus d'expérimentation est moins important dans le cas d'un fonctionnement asynchrone car on exclut une action immédiate de l'opérateur sur les processus en cours. L'exécution de ces deux groupes de processus peut être alors séquentielle ou parallèle suivant uniquement les besoins de partage en ressource processeur.

3. Analyse des ressources existantes

Nous décrivons ici les ressources matérielles et systèmes disponibles en vue d'une implantation des processus introduits jusque là. Ces ressources ne peuvent pas prétendre satisfaire d'une manière idéale l'ensemble des processus. Nous jugeons néanmoins qu'elles constituent un environnement intéressant pour une première mise en oeuvre et surtout pour une maîtrise des mécanismes de base de l'expérimentation temps réel. Le matériel existant nous a guidé dans :
- une étude d'un nombre maximum de fonctions par une étude des contextes des processus et des protocoles de communication,
- une implantation réelle.

3.1. Ressources processeur

Les ressources proviennent essentiellement d'un système LSI 11/02 et d'un processeur vectoriel AP 120B. L'utilisation d'un processeur rapide tel que le processeur vectoriel nous semble une condition nécessaire pour conduire le type d'application actuel. Nous lui attribuerons essentiellement l'exécution des algorithmes relatifs aux modules de simulation donc au processus de simulation mécanique. Une

présentation plus détaillée de ce processeur est faite au chapître[].

Le système LSI 11, outre la facilité de mise en oeuvre offre une souplesse sur la connexion d'organes périphériques et l'ajout de ressources au moyen de cartes périphériques disponibles sur le marché (cartes horloges, convertisseurs,...). Le système d'exploitation RT11SJ (Real Time - Single Job) est un système mono-utilisateur permettant la programmation de certains processus rapides. L'utilisation d'un système mono-tâche disposant de mécanismes rapides d'interuption présente pour nous plus d'avantages au niveau du contrôle temporel des processus qu'un système multitâches (ex RSX) qui fait intervenir le temps d'exécution du "scheduler" , de la gestion du basculement des tâches et de la gestion de mémoire virtuelle attribuée aux tâches.

Les ressources suivantes sont centralisées au niveau du système LSI 11 :

- carte interface avec le processeur vectoriel.
  Cette carte permet la communication avec les deux processeurs en assurant la conversion des formats des bus (QBUS côté LSI, compatibilité UNIBUS du processeur vectoriel). Cette liaison permet d'adresser les registres de commande d'une autre interface située au niveau du processeur vectoriel et d'effectuer ainsi les commandes suivantes:
  - commandes de transfert de données de mémoire à mémoire en mode programmé ou DMA,
  - commandes de chargement des registres internes de l'AP120 par le LSI,
  - passage des données en mode DMA,

- carte horloge programmable capable de produire une IT périodique sur le processeur

- carte convertisseur disposant de 8 convertisseurs D/A et 4 convertisseurs A/D. L'adressage des convertisseurs est multiplexée et ne permet pas de lancer des conversions parallèles. La vitesse d'une conversion est en moyenne de 40 us. La carte dispose d'un mécanisme programmable d'interruption du processeur en fin de conversion.

- carte interface série gérant les entrées-sorties sur la console graphique de dialogue et une imprimante à caractère.

- carte contrôleur de disques gérant les entrées-sorties sur disques en mode DMA. Le contrôle DMA peut être programmé directement à partir de primitives systèmes permettant une vitesse moyenne de 30 ms pour un transfert d'un bloc de 512 octets.

3.2. Mise en ouvre du processeur vectoriel

Ce processeur ne supporte pas lui-même un système d'exploitation, mais un outil de développement est disponible sur le processeur hôte LSI. Un pilotage "général" des processus (chargement dynamique du code, exécution de code, transfert de données,...) éxécutés sur ce processeur est possible par le processeur hôte à partir de primitives évoluées utilisant les registres situés sur la seconde carte interface. Néanmoins ce mode ne convient pas à un contrôle temporel efficace des processus tant au niveau du processeur vectoriel que du processeur hôte (prise en compte du temps d'exécution des primitives, du temps de chargement du contexte des processus dans le processeur vectoriel, interférence possible sur l'utilisation des ressources de l'interface par les primitives et les processus) et ne prend pas en compte la particularité d'exécution de processus cycliques qui ne nécessite pas un rechargement systématique de leur contexte.

Nous nous passerons dans ce cas de l'utilisation de ces primitives, en raison de :

- l'utilisation d'un processus cyclique ayant un code exécutable non variable,
- le chargement préalable du code dans une phase d'initialisation du système,
- synchronisation au moyen d'instructions qui gèrent directement les registres de l'interface. Ces registres apparaissent du côté du processeur hôte comme des adresses absolues dans le champ d'adressage des périphériques, et comme des noms symboliques dans le langage d'assemblage du côté processeur vectoriel.

3.3. Unité de visualisation temps-réel

Elle est constituée essentiellement d'un générateur de vecteurs rapide (temps de cycle pour la génération d'un point = 1 nanosec.) et d'un écran à balayage cavalier. L'absence d'un processeur à ce niveau nous contraint à utiliser en partie le processeur vectoriel à faire le contrôle direct du générateur de vecteur : commande d'affichage de vecteur, luminosité, entretien de l'image sur l'écran.

Malgré les contraintes du "temps réel" ce fonctionnement a été rendu possible grâce à l'utilisation d'une interface parallèle (interface IOP-16) capable de gérer un transfert DMA des commandes d'affichage entre le processeur vectoriel et un périphérique simple tel le générateur de vecteurs. L'échantillon visuel est ici une liste de visualisation formé d'une succession de commandes pour le générateur de vecteurs. Le processus de visualisation gère le transfert DMA de l'échantillon visuel produit via l'IOP-16. Nous disons que cet échantillon est produit quand il contient l'ensemble des commandes

produit par l'éxécution de l'ensemble des modules de visualisation.

La gestion de la liste de visualisation peut être faite de plusieurs manières :
- gestion par simple ou double buffer,
- remplissage d'une liste pendant son transfert suivant les vitesses de ces deux actions, ...

Contrainte de visualisation

Le temps séparant le début d'affichage de deux échantillons visuels différents doit être rigoureusement constant.
Dans le cas où ce temps d'affichage est très court par rapport à ce temps constant, un entretien de l'image sur l'écran peut être nécessaire. Ce cas nécessite le concours du processeur en dehors de sa phase calcul pour la simulation mécanique.
Dans les paragraphes qui suivent nous verrons plus précisément :
- les moyens qui permettent à l'utilisateur de contrôler la visualisation,
- les moyens utilisés pour synchroniser la simulation mécanique, la visualisation et l'entretien de l'image.

4. Implantation

Nous avons attribué la priorité à l'implantation de processus dont le fonctionnement entre dans la classe des fonctionnements en mode synchrone. Cela a permis de satisfaire l'objectif de base qui est l'expérimentation temps réel des objets au moyen d'un bouclage geste-vue. Cette faisabilité de l'implantation des fonctions temps réel va permettre la création de ressources nécessaires à un fonctionnement en mode "asynchrone" tel que :
. l' utilisation des fichiers-gestes créés en temps réel,
. l' utilisation du même code optimisé dans la simulation mécanique pour réduire les vitesses de simulation d'objets complexes lors d'un fonctionnement en mode "asynchrone".

Nous faisons ici une étude approfondie des problèmes de synchronisation et des contextés des processus suivants:
. acquisition temps réel du geste,
. simulation mécanique,
. enregistrement temps-réel du geste,
. lecture temps-réel de fichier geste,
. visualisation temps réel.

Malgré l'absence d'un langage (de "haut-niveau" ou de "bas-niveau") et de primitives système qui permettraient de définir explicitement les processus, nous nous sommes attachés jusqu'ici à l'utilisation de ce

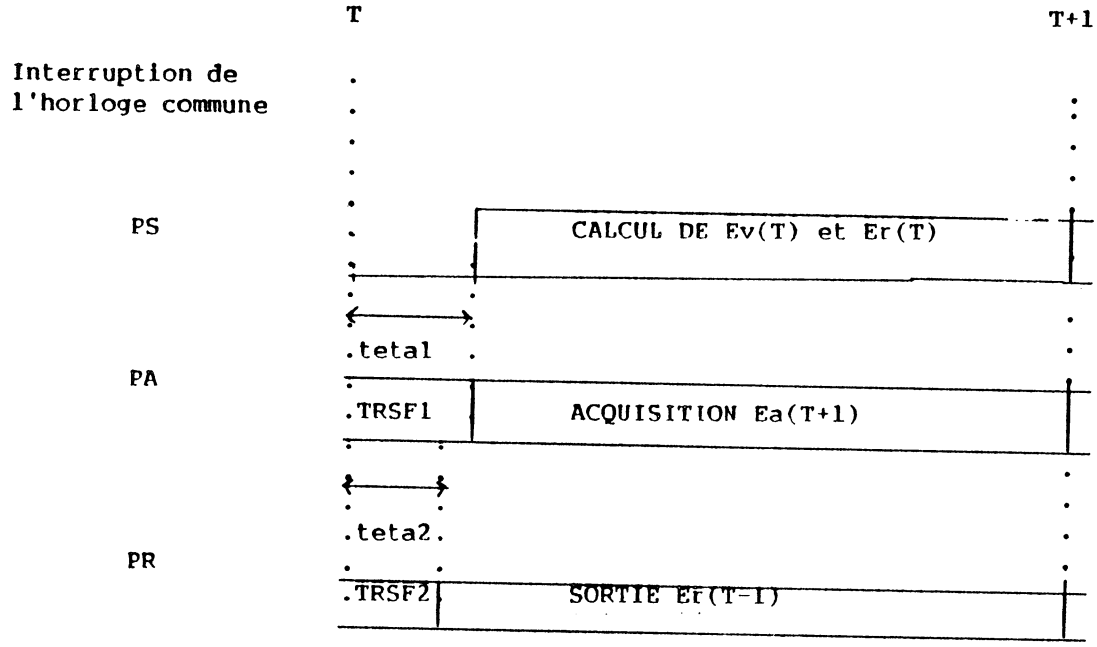
dernier terme qui s'entoure d'un vocabulaire qui décrit assez bien les problèmes que nous avons et qui concerne le parallélisme, la communication, la synchronisation,... Nous essayons dans notre cas d'aboutir à une réduction de ces problèmes qui va nous permettre une implantation " à la main" des processus au moyen des ressources physiques disponibles (utilisation de variables globales, mécanismes d'IT, transfert programmé ou en DMA entre les deux mémoires locales disponibles).

4.1. Communication acquisition geste - simulation mécanique, et restitution geste

Il s'agit ici de la communication entre les trois processus suivant :
- processus faisant l'acquisition d'un échantillon de geste à fréquence constante,
- un processus de traitement utilisant le dernier échantillon de geste et faisant le calcul d'un échantillon de retour d'effort,
- un processus de restitution de ce dernier échantillon sur les transducteurs physiques à la même vitesse que l'acquisition.
Si on fait abstraction de la vitesse des processus, les communications des échantillons reviennent au modèle du producteur-consommateur avec un nombre de message égal à 1.

Pour permettre les contrôles sur les vitesses d'exécution des processus et éviter les pertes d'échantillons, nous utilisons ici le mécanisme de synchronisation par une horloge commune (évoqué à la présentation générale des processus) où on aura une exécution de chaque processus dans un même intervalle de temps qui est la période de cette horloge. Nous utilisons ici la carte horloge-programmable disponible sur le LSI 11.

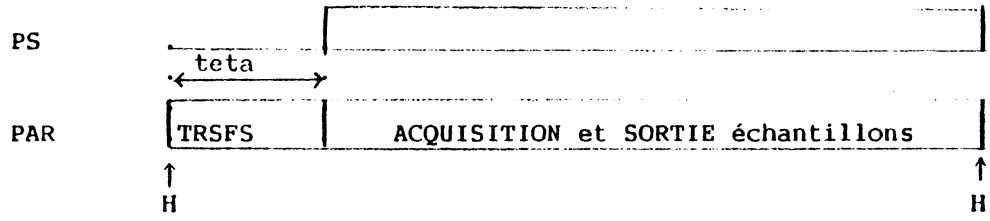
La communication des échantillons entre les processus ne pose aucun problème dans ce cas si la longueur du message (échantillon) est de 1 mot mémoire (sinon l'exclusion mutuelle entre une lecture et une écriture d'une unité de mémoire). Le cas échéant il faut définir les instants précis de communication ou un fonctionnement en double-buffer du message pour éviter les attentes dus à l'exclusion sur l'accès aux échantillons.
Si nous nommons respectivement par PA, PS, PR les processus acquisition geste, processus simulation et processus retour d'effort, nous voyons sur les schémas suivants une illustration de la communication adoptée entre ces derniers. Nous considerons ici que :
- un échantillon est produit par PA quand il a été tranferé (par DMA) dans la mémoire locale à PS,
- un échantillon est produit par PS quand il a fini son écriture dans sa mémoire locale.
- la synchronisation est faite à partir des interruptions de l'horloge programmable.



Ea = échantillon acquis
Er = échantillon retour
TRSF1 = Transfert Ea(T) vers MEM(PS)
TRSF2 = Transfert de Er(T-1) vers MEM(PR)

La référence de temps utilisée pour les échantillons est celle du temps discrétisé au niveau du processus de simulation. Les décalage d'échantillons sont dûs au parallélisme entre les processus.
L'exclusion entre l'exécution des processus évite une gestion en double buffer au niveau des tampons de transfert des échantillons. Les temps de transfert sont très minimes (transfert DMA d'une dizaine de mots) par rapport à la période de l'horloge, nous supposons donc qu'ils ne pénalisent pas l'exécution du processus de simulation.

Dans le but de réduire le nombre de processus en jeu, nous nous sommes aperçu que l'ensemble des fonctions effectuées par PA et PR peuvent s'effectuer séquentiellement dans un temps permettant de respecter la communication adoptée. On peut atteindre en effet une fréquence de 1KHZ par rapport à l'exécution de ces fonctions, nous les faisons alors exécuter par un même processus PAR.
Le diagramme temporel ci-dessus montre les fonctions du processus PAR qui rassemblent ceux des processus de PA et PR.

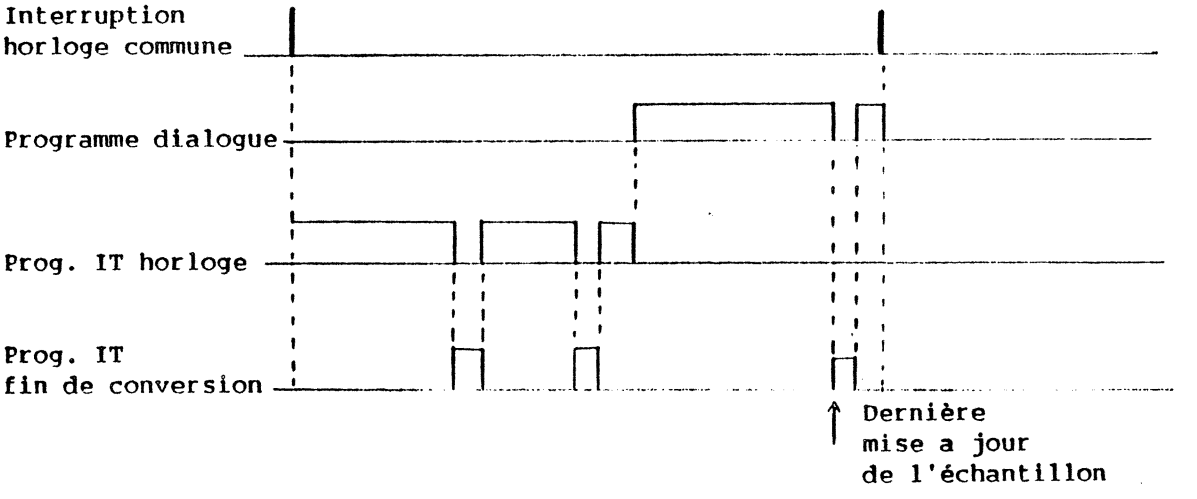


TRSFS rassemble TRSF1 et TRSF2
teta = teta1 + teta2

Pour décrire le schéma du code du processus PAR nous faisons intervenir ici un processus fils élémentaire qui fait l'acquisition sur un convertisseur donné sur un signal de fin de conversion, évitant ainsi le temps d'attente à chaque conversion. La sortie des échantillons sur les convertisseurs ne nécessite pas de temps d'attente. Le code du processus PAR se traduit alors par les 2 schémas Prog1 et Prog2 relatifs au programme d'interruption par l'horloge commune de synchronisation et au programme d'interruption par une fin de conversion.

```
Schema Prog1

Transfert_LSI_AP (echantillon_geste);
Transfert_AP_LSI (echantillon_retour_effort);
numvoie <- premier_voie;
Demarrer_conversion (numvoie);
sortie_conversion (echantillon_retour_effort);

Finschema

Schema Prog2

valeur_convertie <- convertisseur(numvoie);
mis_a_jour (echantillon_geste, valeur_convertie);
si non_dernier_voie alors
   numvoie <- suivant_voie;
   Demarrer_conversion (numvoie);

Finschema
```

Le diagramme temporel qui regroupe les tâches du processeur hôte est le suivant :



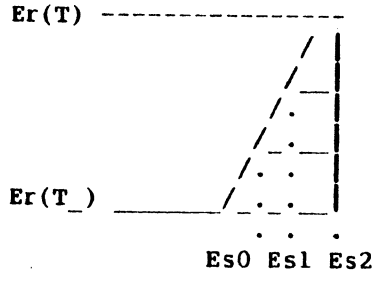
4.2. Suréchantillonnage du geste

Il s'agit ici d'introduire dans l'exécution du processus PS l'exécution de n pas de simulation pour une exécution du processus PAR et une exéction du processus PR, en respectant le même mécanisme de synchronisation. Nous pensons que ceci est indispensable pour la simulation de structure présentant des paramètres de raideur élevée dont la plage qui permet la convergence de la série des variables positions solutions des équations du modèle mécanique est fonction de la vitesse de simulation (cf).
L'échantillon de retour d'effort lu par PAR est produit par le dernier pas de simulation et le calcul des modules de visualisation intervient après celui-ci.
Nous appelerons PSN le processus correspondant au fonctionnement général avec le nombre n de pas de simulation >= 1. La valeur n reste constant pour une expérimentation donnée, elle est facilement déterminée en connaissant le temps d'éxécution d'un pas de simulation. Ce temps est fixe pour la structure du code du processus PSN que nous avons adopté. Ceci fait apparaître notamment :

. une réduction des instructions conditionnelles,

. une utilisation d'instruction conditionnelle à deux branchements et exécution du même nombre d'instructions pour chaque branchement.

L'existance de deux fréquences d'échantillonnage (basse fréquence basse pour l'acquisition du geste et haute fréquence pour la simulation mécanique) introduit une distorsion sur le modèle de simulation. Il nous a fallu introduire un filtre d'adaptation de fréquence d'échantillonnage entre l'acquisition de l'échantillon gestuel et son utilisation par le processus de simulation.
Nous avons adopté ici un filtrage simple qui est l'interpolation linéaire entre les échantillons de geste.



Le filtrage ne joue pas sur l'exécution du processus PAR. Le schéma décrivant le code du processus PSN est alors le suivant :

Schema PSN

```
PSN:                pas_d_interpolation <- (ErT - ErT_) / n;
                    ErT_ <- ErT

Surechantillonnage: Er_simulation <- Er_simulation + pas_d_interpolation;

                    Calcul_modules_mecaniques

                    dec (n)
                    si n>o alors aller a Surechantillonnage
finsurechant:
                    calcul_modules_de_visualisation
                    aller a PSN
```

Finschema

Remarque

Er_simulation indique ici l'échantillon utilisé pour la boucle de suréchantillonnage.
En début d'exécution du processus PSN, ErT et ErT_ indiquent les échantillons acquis par la dernière exécution et l'avant dernière exécution de PAR.

4.3. Processus enregistrement des gestes

4.3.1. Présentation

Ce processus assure la mémorisation des échantillons de geste sur des fichiers-geste, de façon temps réel, c'est à dire à une vitesse moyenne équivalente à la vitesse d'acquisition des échantillons.

La mémorisation est répartie sur deux fichiers correspondant aux deux canaux (II 2.4.1) pouvant être mis en jeu. La répartition en canal est obtenu à partir des arguments définis par l'utilisateur sur les relations Transducteur-Voies physiques et Transducteur-Canal. La séparation en deux fichiers doit pouvoir faciliter des traitements autonomes et une utilisation en exclusion de ces fichiers.

La vitesse élevée nécessaire pour l'enregistrement sur fichier nous amène à l'utilisation directe des primitives du système d'exploitation pour la gestion d'E/S sur disque, à la place des fonctions standards du compilateur. Ces primitives permettent un temps d'initialisation d'un transfert mémoire - disque de 1.75 ms et un temps de transfert maximum de 40 ms pour une taille de buffer de 512 octets, correspondant à la taille d'une trentaine d'échantillons de geste acquis.

Dans un fonctionnement à double-buffer de 512 octets, l'exécution du transfert sur chacun des fichiers peut être parallélisé avec celui des trente acquisitions permettant de remplir un buffer, moyennant une fréquence d'acquisition inférieure à 300 hertz.
Ces hypothèses sur la vitesse et la gestion des buffers vont nous permettre de résoudre :
. le problème d'exclusion mutuelle sur l'accès aux échantillons par le fonctionnement en double buffer,
. le problème de synchronisation avec les processus existants par l'intégration dans le code du processus PAR défini dans le paragraphe précédent la gestion des double-buffer et l'initialisation des transferts.

Remarques

La fréquence de transfert physique sur disque peut être accrue (jusqu'à 1 Khz), pour une taille de buffer optimale de 4096 octets. Elle est néanmoins limitée par la fréquence d'exécution du processus PAR qui fait l'acquisition des échantillons. En effet le temps d'exécution de l'ensemble des fonctions assurées par le processus PAR n'est pas négligeable car il limite cette fréquence à 300 hertz. La taille de 512 que nous avons adoptée correspond alors à une taille optimum, il répond par ailleurs à un souci majeur pour l'optimisation de place mémoire utilisée.

4.3.2. Structure de fichier

Nous avons choisi un enregistrement fixe pour le fichier-geste correspondant au nombre de voies maximum par canal pour faciliter d'une part la préparation des buffers d'E/S et d'autre part pour faciliter la lecture ultérieure des fichiers.

Pour éviter de mémoriser une succession d'enregistrements identiques, nous introduisons un champ compteur CPT pour chaque enregistrement :
si pour toute voie V(t) appartenant à un canal on a :
| V(t-1)-Vt | < seuil
alors CPT(t-1) <- CPT(t-1) +1
t réfère ici au contenu des variables du processus pour sa t-ieme exécution.

4.3.3. Contexte du processus PAR

Nous présentons ici la structure des données et le schéma général du code du processus PAR qui englobe les fonctions : acquisition d'échantillons, retour d'échantillons et enregistrement d'échantillons sur fichiers-gestes.
Nous utilisons pour cela les premiers schémas Prog1 et Prog2 et nous les complétons pour la fonction d'enregistrement sur deux fichiers-gestes relatifs à l'utilisation simultanée de deux canaux.

A) Structure de données utilisées

1- Variables tableaux d'entiers:

CANVOIES(nmax)
nmax indique le nombre maximum de voies physiques en entrée pour un canal donné. Nmax ici est égal a 8.
CANVOIE(i) = numéro du canal (1 ou 2) pour la voie i
= -1 si la voie i n'est pas associée à un canal.
BUFCANA1(256)
BUFCAN11(256)
BUFCANA2(256)
BUFCAN21(256)
désignent les 2 double-buffers d'E/S pour chaque canal Chaque buffer est constitué par une succession de 28 enregistrements et de 4 mots non significatifs avec la structure des enregistrements suivante :
valeur-voie-0,...,valeur-voie-nmax,CPT

BOOLMODIF(2)
indique pour chaque canal si l'ensemble des voies courantes lues est identique à l'ensemble lu à l'échantillonnage

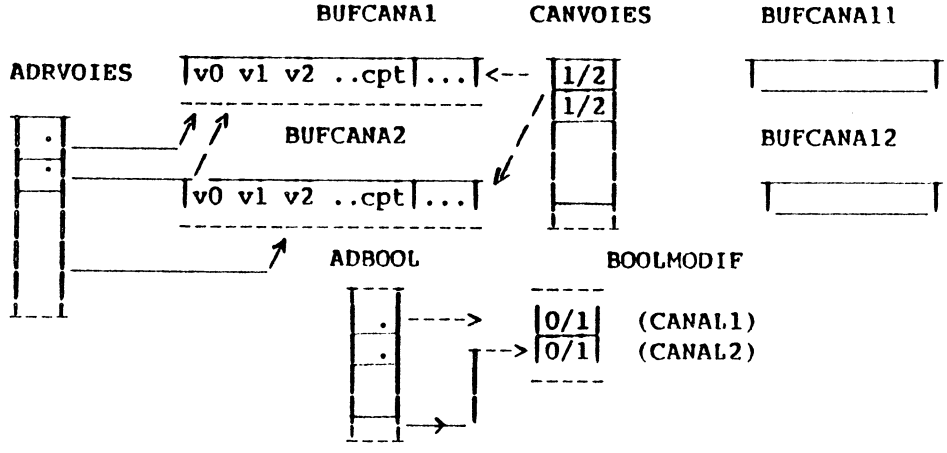
précédent.

ADBOOL(nmax)
pointeurs sur une entrée dans BOOLMODIF avec ADBOOL(i) = @BOOLMODIF(j) si la voie i est associée au canal j. Cet adressage facilite la mise à jour de BOOLMODIF à partir des comparaisons sur les valeurs des voies.

VOIESCNV(nmax)
contient l'ensemble des valeurs courantes des voies de conversion en entrée (num 0 à nmax-1).

ADRVOIES(nmax+2)
pointent sur le champ valeur-voie-"num" représentant la voie dont le numéro est l'indice num du tableau. Cet adressage est nécessaire à l'écriture des valeurs de voie dans les buffers d'E/S courants. ADRVOIES(nmax+i) pointe sur le champ CPT correspondant au canal i.

IBUF1
IBUF2
indiquent la position de l'enregistrement courant pour chaque buffer
IBLOC1
IBLOC2
indiquent la position du bloc courant à écrire pour chaque fichier-ge
BASCUL1
BASCUL2 indiquent les buffers courants de travail pour chaque canal
BLOCMAX1
BLOCMAX2 indiquent la taille en nombre de blocs de chaque fichier-ge



Nous complétons le schéma Progl (décrivant le programme d'interruption de l'horloge commune (6.1)) par la séquence suivante :

```
Pour chaque canal faire
     si IBLOC <> BLOCMAX alors
        si IBUF = 28 alors
           init_dma_disk (IBLOC, BASCUL)
           inc (IBLOC)
           BASCUL <- non BASCUL
           IBUF <- 0
           mis_a_jour_ADRVOIES(IBUF, BASCUL, CANVOIES);
        finsi
     sinon
       fermer_fichier_geste;
     finsi
Finpour
```

Le schéma Progl est complété par la séquence suivante :

```
@ADRVOIES(numvoie) <- VOIESCNV(numvoie);
si difference(@ADRVOIES(numvoie), VOIESCNV(numvoie)) > seuil alors
   mis_a_jour_BOOLMODIF(ADBOOL,numvoie);
finsi
si numvoie > nmax alors
   Pour chaque canal faire
        si non BOOLMODIF alors
           CPT <- CPT+1
        sinon
            inc (IBUF);
            mis_a_jour_ADRVOIES(IBUF, BASCUL, CANVOIES);
            CPT <- 1;
        finsi
   Finpour
finsi
```

Remarques

Nous faisons ici la conversion sur les "nmax" convertisseurs existants. Les 3 fonctions premier_voie, suivant_voie, dernier_voie introduites dans les schémas initiaux de Progl et Prog2 se traduisent donc par :

lère fonction: premier_voie <- 1;
2ème fonction: suivant_voie <- numvoie+1;
3ème fonction: dernier_voie <- numvoie=nmax;

4.4. Processus de lecture de fichier geste

La fonction de lecture se substitue à la fonction d'acquisition des échantillons gestuels par la constitution de ces échantillons au moyen d'une lecture sur les fichiers gestes.
Nous pouvons considérer le fonctionnement en lecture des gestes comme quasiment symétrique de celui de l'enregistrement des gestes, c'est à dire que nous gérons deux double-buffers fonctionnant en bascule pour la lecture simultanée d'au maximum deux fichiers geste. Ce qui permet :
- d'adopter une structure du programme similaire,
- d'utiliser la même structure de données.

Par ailleurs la lecture des fichiers-gestes peut s'effectuer en temps réel dans la mesure où le calcul des temps d'accès sur disque au niveau de l'écriture sur les fichiers restent valables pour la lecture.

Etant donné les hypothèses faites d'une part sur l'exclusion entre les fonctions lecture et enregistrement, d'autre part sur la vitesse de lecture, nous avons pensé intégrer le code d'exécution de la fonction de lecture dans le code du processus PAR. En considérant que le temps utile à la gestion des tampons d'E/S est identique pour la lecture et l'écriture, le processeur est moins chargé dans le mode lecture, en raison de la suppression des acquisitions sur les convertisseurs et de la restitution du retour d'effort. Cela nous a permis d'obtenir une fréquence d'exécution du processus PAR supérieure à 300 HZ.

4.4.1. Constitution de l'échantillon de geste

L'échantillon de geste est obtenu par l'union des voies apparaissant dans les deux enregistrements courants dans les buffers d'E/S. L'interprétation des voies utiles dans chaque enregistrement est faite à partir des arguments qui définissent les canaux et les transducteurs virtuels en jeu.
La constitution d'un échantillon de geste est accompagné de la décrémentation du champ CPT dans l'enregistrement courant. Le passage sur l'enregistrement suivant se fait quand ce dernier atteint la valeur 0.

4.4.2. Schéma de programme

Nous décrivons ici le complément nécessaire au schéma décrivant le code du processus PAR qui inclut la lecture sur fichier en temps réel. Nous utilisons dans ce cas la même structure de données définie pour le contexte précédent du processus PAR. La symétrie par rapport à la structure du programme dans le mode écriture permet de :

- compléter le schéma de Progl avec la même séquence que Seql dans laquelle la procédure init-dma-disk appelée est une procédure d'initialisation de transfert en lecture.
- remplacer dans le schéma Progl la procédure Demarrer_conversion par une procédure de mise à jour conditionnelle du tableau VOIESCNV à partir des enregistrements courants dans les buffers d'E/S. Cette condition est le passage à zéro du champ CPT dans un de ces enregistrements. Le cas échéant le champ CPT est décrémenté de 1.

Nous faisons figurer ici le traitement relatif à la derniere modification.

```
Pour chaque canal faire
     si CPT = 0 alors
        inc (IBUF);
        mis_a_jour_ADRVOIES(IBUF, BASCUL, CANVOIES);
        mis_a_jour_VOIESCNV(ADRVOIES, CANVOIES);
     sinon
        CPT <- CPT -1;
     finsi
Finpour
```

Remarque

Ce dernier traitement nécessite une initialisation du tableau ADRVOIES permettant de pointer sur un enregistrement fictif de champ CPT nul précédent le premier enregistrement du buffer d'E/S .

4.4.3. Utilisation des fichiers-gestes

Les paragraphes précédents illustrent en grande partie le cas de lecture de fichiers-gestes dans un fonctionnement temps-réel. Cela s'avère indispensable en particuliers pour reproduire une expérience instrumentale. Nous pensons par ailleurs qu'un fonctionnement asynchrone entre les processus oblige à l'utilisation de ces fichiers. Nous avons vu que ce fonctionnement n'impose pas des contraintes fortes sur le temps d'exécution des processus, un autre mécanisme de synchronisation peut être dans ce cas, introduit. Pour réaliser la fonction lecture on peut :
- soit garder la même structure du code exécutable défini dans le processus PAR,
- soit garder une partie de ce code et rajouter des traitements propres à une phase "compositionnelle".

Extension possible

Grâce à la possibilité de lecture temps rèel des fichiers il est envisageable de faire une utilisation combinée du fichier geste et d'un transducteur physique pour produire un échantillon courant dans la boucle d'expérimentation temps réel. L'échantillon est dans ce cas produit par la union des voies lues sur les convertisseurs et les voies contenues dans les enregistrements courants des buffers d'E/S. Ceci peut illuster notamment la situation de "play-back".

4.5. Enregistrement par une caméra

Cette fonction permet la possibilité d'enregistrer les images affichées sur l'écran temps réel au moyen d'une caméra. Certains types d'appareil ne nécessitent pas une synchronisation particulière, mais en général ils requièrent la synchronisation introduite ici consistant à faire l'émission d'un signal carré présentant les caractéristiques suivants :
- une période Tc
- une amplitude Ac
- une durée Dc.

Pour éviter de gérer une horloge spéciale à cette fin nous utilisons l'horloge commune à la synchronisation des processus. Nous supposons que la fréquence de cette horloge sera un multiple de 1/Pc quand la fonction d'enregistrement sur caméra est demandée. L'instant de synchronisation avec cette dernière est alors simplement déterminé au moyen d'un compteur qui sera incrémenté à un instant fixe par rapport au début de la période d'horloge. Ce fonctionnement va nous permettre en partie d'intégrer l'exécution de cette fonction par le processus PAR.

L'amplitude est déterminée par la sortie sur un des convertisseurs D/A de la valeur:
v= 2048 + 2048*Tc/5 pour un convertisseur linéaire 12 bits permettant les sorties -5V à +5V à partir des valeurs 0 à 4096.
La durée est obtenue par le temps d'éxécution d'une boucle d'instructions de sortie de la valeur v sur un convertisseur D/A relié à la caméra. La durée Dc nécessaire ici est très courte par rapport à la période utilisée pour la synchronisation et ne pénalise donc pas le temps d'exécution du processus PAR.

4.6. Synchronisation processus simulation mécanique et processus visualisation temps réel

Une exécution simultanée de ces processus nécessite une exclusion mutuelle sur l'accès à l'échantillon visuel. Cette exclusion est ici résolue par l'utilisation de deux buffers

fonctionnant en bascule.

Le processus de visualisation comprend :
- l'initialisation du transfert DMA de l'échantillon visuel vers l'unité de visualisation par l'intermédiaire de l'interface IOP16,
- le transfert DMA de l'échantillon visuel.
La première fonction monopolise le processeur vectoriel pendant un temps minimum qui est le chargement des registres de l'interface.
Ce fonctionnement permet d'obtenir un parallèlisme presque total sur les processus de simulation mécanique et le processus de visualisation. On dispose alors pour l'affichage de l'image (transfert DMA de l'échantillon visuel) d'un temps équivalent au temps d'exécution du processus de simulation ,c'est à dire de la période d'horloge commune de synchronisation.

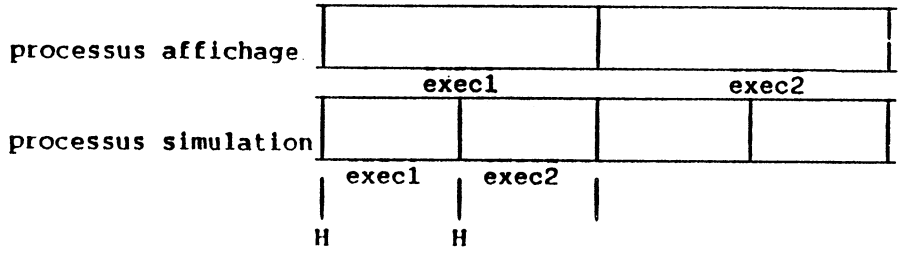
Pour une fréquence d'horloge de 100HZ, le générateur de vecteurs peut afficher une centaine de vecteurs de 200 points (un vecteur = 0.1 ms). L'affichage d'un vecteur nécessite un transfert DMA de 5 mots (coordonnées des points extrêmes, luminosité) à une vitesse de 1MHZ (temps de transfert=0.5us), on s'aperçoit que c'est la vitesse de l'unité de visualisation actuelle qui limite essentiellement le temps d'exécution du processus de visualisation.

Remarque

En raison de la fluctuation du temps d'affichage des images qui est une fonction linéaire de la longueur des vecteurs affichés une situation de débordement par rapport à la période d'horloge commune n'est pas toujours facile à éviter, ce qui risque de faire perdre la linéarité d'affichage. Cette situation est facilement détectée par le test de fin de transfert DMA avant l'initialisation du prochain transfert.
Il est plus judicieux dans ce cas d'avertir l'utilisateur plutôt que . d'arreter l'expérimentation. Cette fonction est assurée par :
- la mise à 1 d'un drapeau au niveau du contexte de processus de visualisation,
- la lecture et réinitialisation de ce drapeau par le processus PAR, et l'émission d'un signal sonore à l'utilisateur si le drapeau est égal à 1.
La communication d'une unité de mémoire permet ici des accès simultanés sans synchronisme particuliers entre les 2 processus.

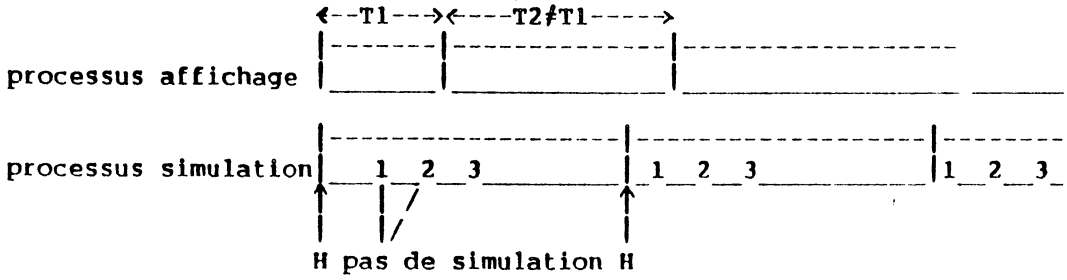
4.6.1. Sous échantillonage visuel

Nous introduisons ici un moyen qui évite ou limite les cas de débordement du temps d'affichage. Il consiste à réduire la fréquence de visualisation, en acceptant de perdre des échantillons dans un rapport constant avec le nombre de pas de simulation effectués. Des phénomènes stroboscopiques peuvent dans ce cas apparaître notamment sur les figures périodiques en raison de la perte des images intermédiaires.
Nous illustrons ici le cas d'un sous échantillonage avec un rapport 1/2.



4.6.2. Situation de la visualisation dans le cas d'un suréchantillonage gestuel.

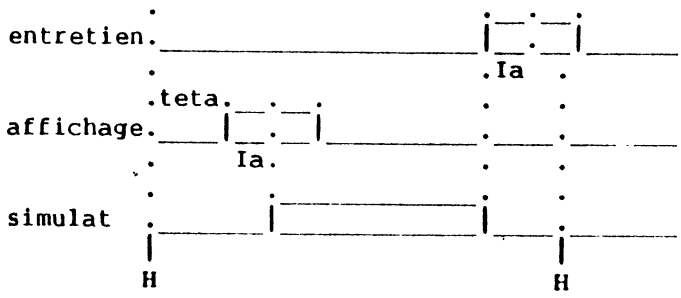
Rappelons que le suréchantillonage cité ici correpond à l'exécution de n (n>1) pas de simulation par période d'horloge commune. Nous n'introduisons pas dans ce cas la possibilité de faire un suréchantillonage visuel (p visualisations pour la période d'horloge avec 1<p<n) car nous pouvons nous retrouver facilement dans le cas de non linéarité sur les images. Dans le cas p=0 ou p=1 on peut avoir les mêmes phénomènes stroboscopiques cités au paragraphe précedent en raison du rapport >1 entre le nombre de pas de simulation et le nombre d'échantillons visuels constitués.
Nous illustrons ci-dessous le cas d'un rapport de 1/2 entre l'affichage et le nombre (=3) de pas de simulation effectués.



4.7. Processus d'entretien de l'image

Dans le cas où le temps d'affichage d'une image est courte par rapport à la période d'horloge, l'utilisateur peut demander un fonctionnement avec réaffichage de l'image pour obtenir une meilleure rémanence sur l'écran. Ce réaffichage est exécuté à chaque fin de l'affichage courant dans l'intervalle de temps séparant l'affichage de l'image suivante et la fin de l'exécution du processus de simulation. Ce fonctionnement risque néanmoins de retarder l'affichage d'un échantillon calculé en raison du dernier réaffichage, nous ramenant au problème de non linéarité sur l'affichage. Nous supposons dans ce cas que les temps de débordement sont réduits dans la mesure où le temps d'affichage est par hypothèse court par rapport au temps d'exécution du processus simulation.

Schema d'implantation qui tient compte du transfert avec le hote et init-affichage



Nous donnons ici le schéma général du programme qui décrit le code pour l'ensemble des processus s'exécutant sur le processeur vectoriel :

```
Schema processus_AP

debut:  si non DMA_image_fait alors
          si non entretien_image alors
                drap_debordement <- vrai;
          finsi
        finsi
        si NB_sous_aff = 0 alors
           attente_fin_DMA;
           init_IOP16_affichage (EvT_);
        finsi

        calcul_n_pas_de_simulation;
```

```
dec(NB_sous_aff);
si NB_sous_aff = 0 alors
   calcul_modules_visualisation (EvT);
finsi

si drap_PAR = 1 alors
   arret_processeur
finsi

si entretien alors
   tant que drap_PAR = 0 faire
        attente_fin_DMA
        init_IOP16_affichage (EvT_);
   fintantque
finsi

EvT_ <- EvT;

aller a debut

finschema
```

Remarques

Dans ce schéma nous avons introduit un contrôle sur le temps d'exécution de la simulation mécanique. Le test de drap_PAR (drapeau de synchronisation qui est mis à jour par le processus PAR) permet de tester si : temps nécessaire à la simulation mécanique + temps de débordement éventuel sur l'affichage > période d'horloge. Dans ce cas nous avons choisi d'arrêter le processeur.
EvT désigne l'échantillon visuel calculé pendant le cycle d'horloge courant.
EvT_ désigne le dernier échantillon visuel calculé avant EvT.
NB_sous_aff désigne le sous-échantillonage visuel. Ici on l'a décrit comme le nombre d'exécution du processus simulation par échantillon visuel produit.

4.8. Les données pour l'environement d'expérimentation

Il s'agit ici de données définies par l'utilisateur qui sont interprétées par le système pour définir :
- les processus à mettre en jeu,
- la communication entre les processus,
- une partie du contexte des processus.

Ces données sont rentrées à partir de trois classes de commandes figurant au niveau de la classe-racine qui sont :
- mise à jour du mode des entrées gestuelles,
- mise à jour du point de vue de l'expérimentation,
- mise à jour du mode des sorties visuelles.

La définition du mode des entrées gestuelles permet de connaître le fonctionnement pour chaque canal qui est un des suivants :
- direct,:lecture sur les voies physiques de conversion (utilisation de transducteurs physiques),
- restitution : lecture de fichier-geste,
- direct mémorisé : enregistrement sur fichier des voies d'un canal fonctionnant en direct.

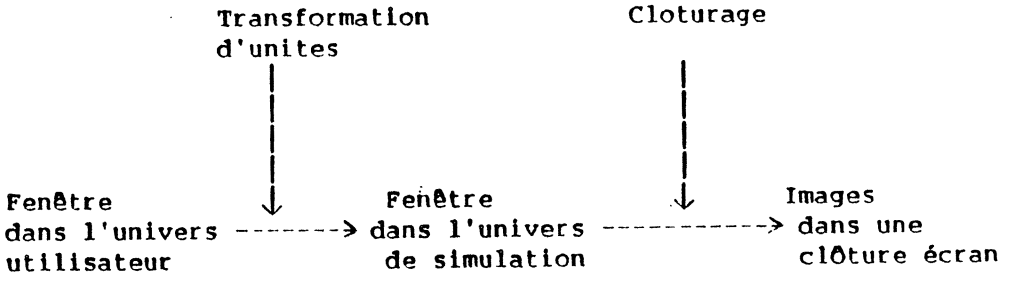
Les points de vue de l'expérimentation regroupent les trois points suivants :

- le point de vue mécanique :

Il décrit le système d'unités utilisé par l'utilisateur pour définir les entités suivantes : masse, longueur, force, raideur, frottement, temps. Ce système est décrit dans le système MKSA Cette description permet de déterminer les coefficients à appliquer sur les paramètres mécaniques et géométriques de l'utilisateur pour obtenir les paramètres de l'univers de simulation.
Nous décrivons à l'annexe l'ensemble des équations utilisées pour cette transformation. Ces équations montrent que des constantes de temps sont pris en compte par les paramètres de simulation et il est important de fonctionner avec une fréquence élevée pour avoir la plus grande plage de validité sur les paramètres. L'utilisateur définit par ailleurs les fréquences d'échantillonage gestuel et d'échantillonnage visuel. A ce titre il définit les trois fréquences suivantes :

. f1, correspondant à la vitesse commune des acquisitions gestuelles et de la restitution du retour d'effort. Cette fréquence est celle de l'horloge commune utilisée pour la synchronisation des processus.
. f2, correspondant à la vitesse de simulation. Le rapport f2/f1 définit alors le surechantillonage gestuel. La période 1/f2 est utilisée comme unité de temps servant dans les équations citées plus haut pour la transformation des paramètres.
. f3, correspondant à la vitesse de visualisation. Le rapport f1/f3 définit le sous-échantillonage à faire pour la visualisation.

- le point de vue spatial :
Nous définissons ainsi l'espace à voir dans l'univers des objets, ainsi que le cadre d'affichage. Ce cadrage est pris en charge par les modules de visualisation et passe par les notions "classiques" de fenêtre et clôture. Le schéma théorique permettant de faire ce cadrage est le suivant :



Les raisons suivantes nous a entrainé à ne pas avoir implanté pour le moment cette procedure de cadrage :
. la définition d'une fenêtre utilisateur adéquate nécessite un premier apprentissage dans l'expérimentation de la dynamique des objets, dans la mesure où les manipulations gestuelles peuvent faire varier de manière importante l'espace balayé par des éléments de ces objets. Cela doit déboucher sur une notion de fenêtre "dynamique" qui permettra à l'affichage de suivre un déplacement continu sur les objets. Cela n'est pas à priori indispensable pour des déplacements périodiques.
. l'introduction des algorithmes de clôturage coûtent cher en temps (notamment en raison des instructions de tests d'inclusion) et nous ne voulons pas répercuter ce temps dans une première évaluation sur la simulation mécanique.

Nous avons introduit ici une procédure de cadrage plus intuitive où les coordonnées écran résultent d' une opération d'homotéthie-translation dont les coefficients sont déduits à partir des variables positions initiales dans les modules matériels.

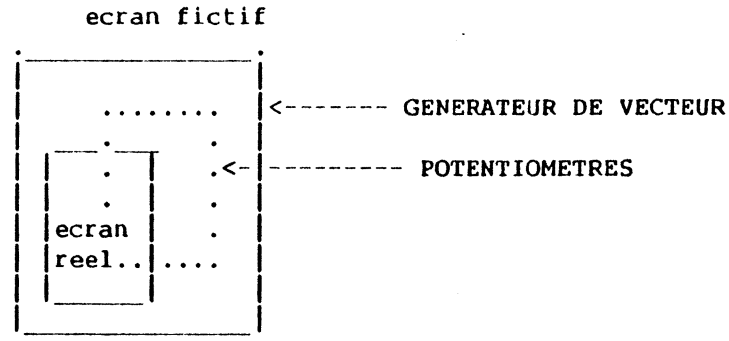
Ceci permet d'avoir une fenêtre infinie et une clôture infinie, mais les paramètres de l'opération définie précédemment permet d'approcher l'espace résultant de la transformation à une partie de cette clôture qui est celle de

l'écran.
L'utilisation d'un écran disposant de potentiomètres pour faire déplacer la plage de tension en entrée (en X et Y), combinée à l'utilisation d'un générateur de vecteurs de gain supérieur à cette plage nous permet de faire "varier" manuellement la clôture représentant l'écran.

gain du générateur ---------> écran fictif

potentiometres ---------> déplacement de l'écran réel dans l'écran fictif

L'affichage dans l'écran fictif ne crée pas de perturbation sur l'image affichée dans l'écran réel.



4.9. Synchronisation des processus dans le mode "asynchrone"

4.9.1. Présentation

Nous abordons ici la description de la synchronisation des processus suivants :
- lecture de fichiers gestes,
- simulation mécanique,
- enregistrement d'échantilons visuels.

Le mécanisme introduit ici tient compte des conditions suivantes :
- vitesse variable de chaque processus,
- exécution en parallèle du processus dialogue,
- nécessité d'optimisation des temps d'exécution : ex. utilisation du principe de lecture en double-buffer (un buffer contient n enregistrements d'un fichier geste) pour optimiser les temps d'E/S sur fichiers. Ce qui n'exclut pas l'utilisation du code dans le contexte des processus fonctionnant en temps-réel.

Nous étudions ici un schéma général d'implantation inspiré du modèle du consommateur-producteur, en utilisant les ressources matérielles de synchronisation disponibles :
. interruption du hôte par l'horloge programmable,
. interruption du hôte par le processeur vectoriel,
. interruption en fin de transfert DMA lecture sur fichier disque (exécution d'une "completion routine").
. possibilité de mise à 1 automatique d'un drapeau du côté processeur vectoriel à la fin d'un transfert DMA.

4.9.2. Communication d'échantillons de geste

En raison de l'absence de mécanisme cablé tel que l'instruction TAS il n'existe pas ici de solution apparent qui assure le mécanisme d'exclusion mutuelle entre deux processeurs qui veulent faire une gestion commune des indices Nplein et Nvide dans le tableau des échantillons [26 page 20,38]. Nous supposons alors ici que :
- la communication d'un échantillon se fait sur une demande de processeur vectoriel par interruption du hôte, sous certaines conditions à chaque fin de simulation,
- la simulation démarre après le transfert DMA de l'échantillon dont la terminaison est signalé par le drapeau drap_fin_DMA mis par le hôte et rénitialisé par le processeur vectoriel,
- les échantillons prêts sont stockés au niveau du hôte.

4.9.3. Communication d'échantillons visuels

Nous ne pouvons pas avoir un fonctionnement symétrique à la lecture dans la mesure où le processeur vectoriel n'est pas interruptible. D'autre part un échantillon peut être de taille importante et le nombre stockable sera dépendant de cette taille et de la mémoire disponible. Pour minimiser l'occupation mémoire du processeur vectoriel pour raison de coût nous supposons qu'on dispose du côté hôte un tampon (un tampon = m échantillons visuels) qui regroupe les 'chantillons produits. Nous utiliserons ici 2 tampons fonctionnant en bascule pour optimiser le transfert disque. Chaque transfert est précédé d'un test de fin du transfert précédent.

Du fait que la production d'un échantillon coincide avec la demande d'un échantillon de geste, nous pensons que le transfert d'un échantillon visuel peut être commandé par la même interruption du hôte décrit précédemment. La possibilité de produire un nouvel échantillon est signalé par du drapeau Fin_trsf "cablé" après la terminaison d'un transfert. Ce drapeau est réinitialisé par le processeur vectoriel.

4.9.4. Choix de la décomposition en processus

Nous avons choisi de faire une exécution du processus qui assure en séquence la fonction lecture et la fonction enregistrement, à partir de l'interruption de l'horloge programmable du hôte pour les raisons suivantes :
- le nombre d'échantillons lus est égal au nombre d'échantillons produits (à une unité près),
- nous prenons l'hypothèse :
temps moyen d'exécution du processus de simulation = temps moyen de lecture d'un échantillon + temps moyen d'enregistrement d'un enregistrement + temps minimum pour le dialogue. La période de l'horloge choisie doit être proche du temps moyen Tm.

4.9.5. Conséquences par rapport au mode de fonctionnement "synchrone"

Nous prenons le code du processus qui assure la fonction lecture en tenant compte des modifications suivantes :
. le nombre d'échantillons prêts peut être > 1,
. le transfert se fait soit sur demande du processeur vectoriel au moyen d'une interruption, soit après la production d'un échantillon si la demande n'est pas satisfaite,
. la gestion d'écriture d'un nouvel échantillon prêt est inintéruptible,
. le transfert DMA sur disque est précédé d'un test de fin de transfert. test de fin de transfert.

4.9.6. Sch'ma de programme

Nous donnons ici le schéma de principe qui décrit les différentes modifications de la fonction lecture par rapport au mode "synchrone", ainsi que la fonction enregistrement :

```
Schema init;
  Nplein=0;
  Nvide=Lgtable;
  drap_debord<-faux;
  dma_lect<-vrai;
  dma_ecr<-vrai;
  tampon_sortie_plein<-faux;
Finschema
Schéma Prog_IT_AP_LSI
        masquer_IT;
        si Nvide=-1
           alors APfamine <- vrai;
           sinon transfert_DMA_1 (table_echantillons,
                                  Nvide, Nplein);
                 transfert_DMA_1 (drap_fin_DMA_1)
           finsi

        si non tampon_sortie_plein
           alors transfert_DMA_e (Fin_trsf, echantillon_visuel,
                                  tampon_sortie, tampon_sortie_plein);
           sinon AP_bloque <- vrai
           finsi
        demasquer_IT;
Finschéma

Schéma Completion_routine_fin_DMA_disq_lecture
        dma_lect <- vrai
Finschéma
```

Nous faisons les 3 modifications suivantes au niveau du code décrit pour le fonctionnement temps réel (paragraphe 6.3) :

1- la production d'un échantillon geste (appelé ici echantillon_produit) est suivi du test suivant :

```
si APfamine
    alors transfert_DMA (echantillon_produit);
          transfert_DMA (drap_fin_DMA);
    sinon masquer_IT;
          ranger(table_echantillons, echantillon_produit,
                 Nvide, Nplein);
          demasquer_IT;
finsi
```

2- la production d'un échantillon se fait à condition que le tableau d'echantillon ne soit pas plein. Par ailleurs le programme d'interruption teste la terminaison du programme d'interruption précédent. Le programme d'interruption sur l'horloge programmable démarre donc par la séquence suivante :

```
Seq:
si drap_debord
   alors RTI
   sinon drap_debord <- vrai;
finsi

masquer_IT;
si Nplein=-1
   alors demasquer_IT
         aller a enregistrement;
finsi
demasquer_IT

Finseq:
```

3- Le transfert disque - tampon de lecture doit s'assurer de la fin du transfert précédent de la manière suivante :

```
si dma_lect
   alors dma_lect <- faux;
         init_DMA_disk;
finsi
```

Nous donnons ici la séquence correspondante à l'enregistrement de l'échantillon visuel :

```
enregistrement: traitement_eventuel(tampon_sortie);
                si tampon_plein
                   alors si dma_ecr
                            alors dma_ecr <- faux;
                                  init_DMA_disk_e;
                                  basculer(tampon_sortie);
                                  si AP-bloque
                                     alors tranfert_DMA_e(Fin_trsf);
                                           AP_bloque <- faux;
                                  finsi
                         finsi
                finsi

finenr:drap_debord <- faux;
       RTI


Schéma Completion_routine_ecr
        dma_ecr<-vrai;
Finschema
```

CHAPITRE V - IMPLANTATION DES MODULES DE SIMULATION

1. Introduction

Le propos de ce chapitre est l'étude d'une implantation des modules de simulation sur le processeur vectoriel AP120-B de la famille FPS. Cette famille réunit des machines qui sont d'une assez grande universalité et disposent d'une architecture commune facilitant la compatibilité des logiciels. Cette raison explique en partie le choix d'implantation actuelle. Une étude plus globale concernant la "mise en vecteurs" des données est néanmoins nécessaire et ne fait pas intervenir directement ce choix. Nous abordons ce chapitre par une brève présentation de la nature d'un processeur vectoriel et ensuite celle du processeur AP120-B.

2. Présentation d'un processeur vectoriel

Le processeur vectoriel désigne une unité arithmétique rapide reliée au bus d'un ordinateur hôte. Cette unité arithmétique est adaptée aux calculs mettant en oeuvre un nombre élevé d'itérations sur des opérations d'addition ou de multiplication en flottant, et faisant intervenir rarement des calculs "complexes" d'adresses. Les données sont disposées sous forme vectorisée c'est à dire sous forme d'une succession d'éléments subissant les mêmes traitements et dont l'adresse d'un élément découle immédiatement du précédent (ex. par simple une incrémentation d'un registre). Les calculs peuvent être analysés en ces termes dans les applications faisant intervenir le traitement numérique du signal, l'arithmétique matricielle, l'analyse statistique et la simulation numérique.

Le processeur vectoriel peut avoir un fonctionnement parallèle et indépendant du processeur hôte et il accepte généralement le type de jeu d'instructions élémentaires sur une machine classique. Dans un programme en langage d'assemblage, le programmeur peut être amené à gérer les instructions à un niveau plus bas comme la commande des étages d'une structure pipe-line ou la commande d'un parallélisme interne. L'utilisation de procédures "systèmes" au niveau du hôte

peut néanmoins rendre transparent l'utilisation de ce processeur.

CYCLE	OPERATIONS	RESULTAT
1	A*B	RESu
2	E*F	RESv
3	M*N	RESw
4	OPi	A*B
5	OPj	E*F
6	OPk	M*N

Ex: multiplieur à trois étages

Le processeur vectoriel est dans ce cas à distinguer :

a) de la carte (ou boîtier) accélérateur flottant qui est gérée par un même séquenceur que celui de la machine où elle est installée, et exécute en séquence les instructions d'opérations en flottant sous forme cablée ou microprogrammée. Ces instructions sont généralement traitées par un émulateur logiciel en l'absence d'une telle carte.

b) du processeur spécialisé qui intègre des algorithmes (au sens des opérations non élémentaires) sous forme cablée ou microprogrammée.

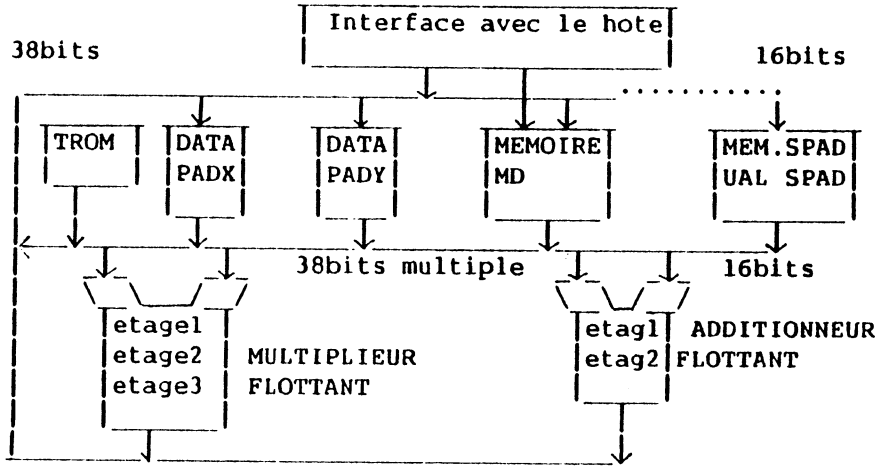
Le choix d'utilisation d'un processeur vectoriel est essentiellement justifié par les performances de calcul de celui-ci à savoir :
- une vitesse d'exécution des opérations arithmétiques pouvant atteindre un facteur de 10 à 1000 relativement à la vitesse d'une machine classique,
- une résolution sur la représentation des nombres en virgule flottante permettant une bonne dynamique de calcul. Cette dynamique est assez mal respectée pour les algorithmes itératifs implantés sur un ordinateur conventionnel, en raison des répercussions d' erreurs du type erreurs de troncature. Il existe pour cela des moyens de rattrapage que nous ne discuterons pas ici (moyens statistiques, algorithmes stochastiques...), et consommant néanmoims du temps machine supplémentaire pour leur mise en oeuvre.

3. Présentation de la configuration actuelle du processeur AP120-B

3.1. Architecture générale

La machine est à architecture à bus multiples permettant un fort parallélisme interne. Ce parallélisme concerne l'exécution des opérations essentielles du traitement vectoriel tels que le contrôle de boucle, la lecture/écriture mémoire, l'opération d'addition et de multiplication sur les réels. Les unités arithmétiques ont une

structure pipe-line et la possibilté par le programmeur de réutiliser directement chaque étage libre permet d'avoir des résultats de calcul tous les 167 ns. Ce temps correspond à un cycle de base de la machine et toute opération s'effectue en un nombre fixe de cycle qui est connu par le programmeur. Une opération ou plusieurs opérations parallèles peuvent être lancés à chaque début de cycle.



3.2. Organisation de la mémoire

Les éléments mémoires sont répartis en plusieurs bancs pour permettre une spécialisation de chaque banc et faciliter le fonctionnement parallèle entre les bancs. Ces bancs regroupent :

a) le banc de mémoire TMROM
C'est une mémoire morte rapide (accès à la donnée en deux cycles de base) et sert au rangement de tables de constantes utilisées dans les fonctions mathématiques et dans l'optimisation de certains algorithmes de base inclus dans la bibliothèque (calcul racine carrée, division, FFT, ...).

b) le banc de mémoire DATAPAD
Il s'agit de 2 blocs de 32 accumulateurs 38 bits(DPX et DPY) à accès rapide, l'accès à un accumulateur est indexable au moyen d'un registre d'index. Les registres de cette mémoire sont reliés directement aux unités arithmétiques et constituent en général des registres de travail ou des registres de mémorisation temporaire d'opérandes.

c)le banc de mémoire centrale MD
Elle représente la principale ressource mémoire et est constituée de mots de 38 bits. Le temps d'accès à la donnée est de 3 cycles de base et une demande d'accès peut être initilisée tous les cycles. Sa taille actuelle est de 8K, elle est extensible par bloc de 8K jusqu'à 64K.

d) mémoire programme PS

La mémoire PS contient des instructions de 64 bits, où chaque instruction contient l'ensemble des champs correspondant à des commandes d'opérations pouvant être exécutés en parallèle. L'incompatibilité au niveau du parallélisme dans un programme utilisateur est détecté au niveau de l'assemblage des programmes.
Dans une instruction donnée, le champ adresse d'une opérande (dans le sens d'une architecture conventionnelle) est remplacé par un champ commandant le chargement d'un registre d'adresse associé au banc de mémoire utilisé. Le chargement effectif de ce registre initialise l'accès à l'opérande. Dans le cas d'un accès à un accumulateur du DATAPAD le contenu de ce registre est additionné au contenu du registre d'index.

3.3. Traitement des entiers

L'unité SPAD regroupe 16 registres d'entiers de 16 bits et une ALU permettant les opérations arithmétiques et logiques sur les entiers chargés dans les registres. Cette unité sert essentiellement à la gestion des compteurs de boucle et le calcul des adresses des opérandes contenues dans les bancs de mémoire MD et TMROM et son fonctionnement peut être parallélisé avec celui des unités de calcul en flottant..

4. Estimations sur le temps de calcul

Quelques évaluations simples nous semblent nécéssaires pour estimer le temps de résolution des équations mis en oeuvre dans les modules de simulation. Nous prenons ici comme référence le temps pris pour une machine LSI 11/02 muni du boîtier accélérateur flottant (FIS) qui nous était disponible. Cette évaluation permettra de connaître les limites de la faisabilité d'une implantation dans le cas d'un mini-ordinateur classique de gamme moyenne, si nous ne voulions pas recourir à un investissement assez important qu' est l'acquisition d'un processeur vectoriel. Pour simplifier l'évaluation nous reprenons les équations du point matériel et de la force de liaison introduites au (I- 2.1.) prises dans un modèle d'objets à une dimension spatiale.
Le calcul de l'abscisse du point matériel est du type :
x(n)=f(n-1) * 1/m + 2*x(n-1) - x(n-2)
Ce calcul nécessite 2 multiplications et 2 additions en considérant que les temps d'exécution d'une soustraction et d'une addition sont identiques.
Le calcul de la force de liaison à une dimension est du type :
f(n)=k*dl + z*dv soit
f(n)=k * (x1(n)-x2(n)-L0)) + z * ((x2(n)-x2(n-1))-(x1(n)-x1(n-1)))
Ce calcul fait intervenir 6 additions et 2 multiplications. Une addition supplémentaire est nécessaire pour faire la sommation des forces dans le cas où plusieurs forces s'appliquent à un même point

matériel.
Dans l'exécution de ces 2 modules interviennent 4 multiplications et 9 additions, et si on prend comme temps moyen d'exécution d'une multiplication 100 us et d'une addition 50 us on obtient un temps approximatif de 1ms. Ceci correspond à une fréquence de boucle de 1khz pour un couple masse-ressort, ou une fréquence de 100Hz pour une "corde" comportant 10 masses. Nous supposons que cette vitesse sera réduite au moins de moitié si on prend en considération :
. une visualisation simplifiée,
. les opérations de lecture/écriture en mémoire centrale,
. la prise en compte d'une simple action gestuelle sans rétroaction pour manipuler l'objet.

Les premières expérimentations qui ont conduit à l'étude du système actuel ont été faites dans les conditions de calcul évoquées précédemment et elles ont abouti à l'implantation du programme ANIMA-I [46] qui a servi à la production des premières images dynamiques [46]. La structure de ce programme implanté sur un LSI-11/02 est présentée ici et nous la faisons suivre de quelques remarques.
Le programme fait intervenir une boucle de simulation qui est obtenue par l'éxécution des instructions élémentaires générées à partir d'un assemblage de macros-appels par le MACRO-ASSEMBLEUR du système RT11-SJ. La séquence des macro-appels est écrite par l'utilisateur au moyen d'un éditeur de programme classique, et elle permet de décrire l'aspect "qualitatif" d'un objet à simuler. Le macro-appel correspond en effet à l'appel d'une "primitive" de simulation qui est similaire au module de simulation. L'aspect "quantitatif" et les relations entre des éléments de l'objet figure au niveau des arguments des macro-appels. Le type de langage de construction d'objets par assemblage a été dans le programme MUSIC V (avec des "primitives" tels que oscillateurs, générateurs d'enveloppes,...) et dans le programme CORDIS (avec les "primitives" cellule vibrante, liaison entre cellules,...)[46].
Le macro-appel défini dans le programme ANIMA-I permet :
. la réservation mémoire des données necessaire pour la "primitive". Cette réservation utilise la directive .CSECT qui permet d'isoler les donneés du code des instructions,
. la mise en séquence du code correspondant à la "primitive",
. le calcul des adresses pour les données manipulées par le code à partir des arguments des macros-appels.

Modification de paramètres mécaniques.
L'équivalence sur les adresses mémoires établie par l'utilisation du mot réservé COMMON du Fortran et de la directive .CSECT du macro-assembleur permet de retrouver et faire la mise à jour des paramètres de type mécanique de l'objet à partir d'une procédure du programme écrite en Fortran. Cette mise à jour est possible après un arrêt de la boucle de simulation, qui est actionné par une frappe

de caractère sur le clavier alphanumérique. L'utilisation du seul processeur LSI 11/02 ne permet pas de traiter parallèlement les 2 tâches compte tenu des contraintes de temps sur la boucle de simulation.

```
BCLSIMUL: sequence_macro_appels

          si carac_entre alors RETOUR (* caractere rentré sous
          aller a BCLSIMUL               interruption *)
FINBCL:

Schéma Principal _animal

Iterer
   BCLSIMUL;
   modif_paramètres;
finiterer

Finschema
```

Remarques générales

La vitesse de la boucle de simulation a permis d'obtenir des images temps réel mais a nécessité une optimisation maximale sur les opérations de transfert mémoire. Ce programme a par ailleurs délivré des résultats révélateurs quand à l'intérêt nécessaire à pousuivre dans la démarche.

En dehors de l'insuffisance en ressource processeur permettant de simuler des objets complexes, les lacunes se situent :

- au niveau d'une interface adéquate entre le programme et l'utilisateur pour la construction des objets. L'utilisation d'un langage informatique comme le "macro-assemblage" impose en effet des contraintes pour des utilisateurs non initiés. Par ailleurs l'absence d'entités "évoluées" sur la définition de l'activité gestuelle (exemple le transducteur virtuel) conduit l'utilisateur à gérer directement les voies physiques des convertisseurs connectés aux organes de manipulation. Notons que l'introduction d'une interface gestuelle gérant ces entités suppose d'introduire un processus parallèle à la boucle de simulation et aurait nécessité soit un autre processeur, soit plus de rapidité pour le processeur central où s'exécute cette boucle,

- au niveau d'une modification "dynamique" des objets. Cette modification d'objet suppose de passer par une modification de la séquence des macro-appels, et nécessite une recompilation de la séquence des macro-appels et une édition de liens complète du programme,

- au niveau de la modification interactive des paramètres de l'objet. Le dialogue à ce niveau est établi essentiellement sous forme de question-réponses et ne permet pas de faire des affectations par défaut ou globales. Ce type de dialogue ne facilite pas la possibilité de faire des expérimentations successives.

Par rapport à cette situation nous pensons que l'utilisation complémentaire d'un processeur vectoriel permet à l'ordinateur hôte de :
- se décharger de l'exécution de la boucle de simulation. Cette situation permet à celui-ci :
  a- d'effectuer la procédure de modification en parallèle au processus de simulation dans le cas où on adopte une modification dynamique des objets.
  b- d'exécuter des tâches complémentaires à la boucle pour des fonctions de mémorisation , tels l'enregistrement d'échantillons gestuels et d'échantillons visuels qui peuvent servir de ressources à une phase compositionnelle.

- d'accroître la complexité des objets à simuler tel le passage des équations du modèle d'objets à une dimension au modèle à deux dimensions. Le calcul sur ce dernier fait intervenir certaines opérations (ex. division, racine carrée) qui deviennent vite très lourds pour une petite machine comme le LSI 11/02.

- d'accroître la multiplicité des éléments en terme d'éléments nécessitant le traitement vectoriel. A ce sujet l'approche que nous avons faite consiste à introduire les modules de simulation nécessitant pour chaque type un calcul répétitif relatif à sa multiplicité. Le traitement de chaque type est associé à une configuration des données sous forme de "vecteurs".

5. Eléments de programmation du processeur vectoriel

5.1. Présentation

Nous évoquons ici quelques aspects sur la programmation du processeur qui vont nous guider dans l'implantation des algorithmes de simulation. Ces aspects font notamment référence à l'utilisation d'un ensemble de procédures optimisées et à la notion de "boucle unique optimisée" qu'il nous semble important d'éclaircir avant de faire cette implantation. Nous disposons en effet au niveau du système d'exploitation d'une bibliothèque de procédures portant sur des vecteurs (ex. Ci = Ai*Bi , C = Ai*Bi ,... avec @Ai = @Ai + constante ...) qui peuvent être utilisés par un appel procédural. Les paramètres sont généralement des entiers représentant des adresses ou compteurs de boucle et sont passés par les registres du

SPAD.

5.2. Optimisation de boucle de calcul

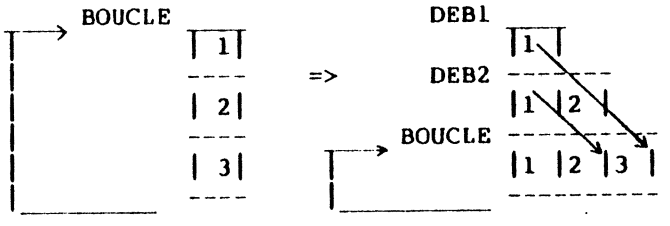
L'utilisation des procédures de bibliothèque ne correspond pas toujours à une utilisation optimum des ressources quand elles ne concordent pas à la particularité de l'algorithme mis en jeu, mais elles se révèlent indispensables pour des problèmes "courants" nécessitant de gros calculs (inverse de matrices, résolution d'équations,...) sans vouloir résoudre les problèmes spécifiques à l'application.
L'écriture d'une boucle qui traite globalement un problème sans fractionner sa résolution au moyen d'appels consécutifs de procédures, peut apporter une optimisation dans l'utilisation des ressources du parallèlisme et dans le nombre des accès mémoire. Une boucle ainsi optimisée nécessite un investisment assez important dans le temps de développement en faveur de l'optimisation du temps d'exécution et il est alors judicieux de trouver un compromis. Pour illustrer ce problème nous prenons ici, sans rentrer au niveau des instructions, un exemple simple qui est le cas du calcul d'une expression
Ei = Ai*Bi + Ci*Di
à partir des deux procédures disponibles :
E1i = Ai*Bi et E2i = Ai + Bi.
Les calculs séparés des E1 et de E2 n'exploitent pas le parallélisme du multiplieur et de l'additionneur et nécessitent des accès mémoires intermédiaires qui sont coûteux en raison des temps d'accès.

Nous introduisons ici quelques recettes dans l'optimisation d'une boucle de calcul, sachant que des méthodes systématiques n'existent pas à ce niveau :

Une première étape consiste à traduire l'algorithme en une séquence d'instructions qui ne comporte pas les opérations en parallèle. Cette étape est analogue à l'écriture d'un programme en langage d'assemblage "classique".

Dans une deuxième étape on se propose de partager la séquence en un certains nombre de blocs qui peuvent être "repliés", et permettre ainsi de réduire la longueur de la boucle par un facteur égal à ce nombre. Ce "repliement" consiste à anticiper sur le calcul des expressions portant sur les indices de boucle suivants grâce à l'utilisation des parallelismes possibles. On arrive généralement à une saturation dans un repliement de quatre blocs correspondant au nombre d'instructions parallélisables (addition, multiplication, accès mémoire, transfert entre registres).

Dans une troisième étape on se pose le problème de la cohérence du calcul entre les blocs dans le démarrage de la boucle en raison du repliement des blocs. Ceci notamment dans le cas où il y a une continuité temporelle dans la séquence initiale (première étape) des instructions (ex séquence d'initialisation d'accès mémoire et lecture de la donnée) ou une relation d'ordre entre les opérations (ex utilisation d'un résultat partiel du bloc i par le bloc i+1). Pour conserver cette continuité il est nécessaire de précéder le démarrage d'un bloc donné par une duplication des blocs de gauche, ce qui aboutit à une structure en "escalier" constituant une "amorce" à la boucle principale. Les schémas suivants permettent d'illuster le principe du repliement :



5.3. Principe d'une boucle optimisée repliée

Il consiste à faire une première utilisation des ressources du parallélisme dans le calcul d'une expression portant sur l'indice de boucle courant grâce à une première décomposition de l'expression en termes dont les calculs sont parallélisables. Nous disons qu'une telle boucle est optimisée et un repliement de la séquence ainsi constituée permet une utilisation maximale des ressources du parallélisme.

5.4. Remarques générales

L'adoption de boucle repliée rend difficile l'utilisation des appels de procédure en raison des ruptures de séquence occasionnés par ceux-ci empêchant le côntrole des ressources du pipe-line et le côntrole des lecture/écritures en mémoire. Notons néanmoins qu'une telle boucle ou une séquence d'appels à des procédures permettent d'aboutir chacune des résultats convenables moyennant une structure de données adaptée. Dans la première méthode, une certaine souplesse peut être obtenue sur les formes de vectorisation des données car celles-ci ne sont pas imposées par la spécification des procédures de bibliothèque. Cette méthode permet en plus d'obtenir une implantation performante et spécialisée pour une application donnée. Par ailleurs elle restreint les possibilités de mise à jour de programme en raison de la difficulté d'"ouvrir" la boucle dans le cas d'une modification de programme.

La méthode par appels de procédures permet par contre de garder une modularité et une facilité de mise à jour. Nous avons adopté cette méthode pour l'implantation des différents types d'équations de liaison, qui se partagent des procédures communes, et où l'utilisation de procédures plus ou moins évoluées en bibliothèque (calcul de division, distance, racine carrée...) nous épargne une tâche de programmation non négligeable consistant à traduire celle ci en une boucle suivant la première méthode. La situation actuelle répond assez favorablement à ce fonctionnement modulaire dans la mesure où il sera nécessaire d'expérimenter de nouvelles caractéristiques au niveau des modules de liaison (exemples : prise en compte d'un frottement du milieu, nouvelles conditions pour les liaisons conditionnelles, ...). Nous nous sommes efforcés à implanter le reste des équations (modules matériels et modules d'entrée) au moyen de boucles repliées ou optimisées pour favoriser notamment la vitesse d'exécution. Nous décrivons quelques exemples de ces programmes en annexe.

6. Etude du contexte du processus de simulation

6.1. Introduction

Nous avons présenté jusque là des aspects de programmation relatifs à la résolution d'équations sur le processeur vectoriel actuel et nous abordons dans les paragraphes suivants :
- les problèmes de la structuration des données,
- la structuration générale du code exécutable pour simuler la structure expérimentable courante.

Il s'agit en effet d'étudier la génération d'un contexte pour le processus de simulation qui comprend le code exécutable du processus et l'ensemble des données dont la structuration facilite le traitement vectoriel. Nous présentons pour cette étude un certain nombre d'outils capable de générer ce contexte et sur lesquels nous avons dû faire un choix.

Ces outils doivent prendre en considération les processeurs sucseptibles de générer ce contexte qui sont le processeur où s'exécute le processus de simulation et le processeur hôte. Le choix qui doit être fait sur les outils doit répondre aux facteurs suivants :

a) rapidité d'exécution du processus de simulation. Ce facteur va déterminer l'architecture du système actuel, c'est à dire la possibilité de respecter une vitesse de boucle d'expérimentation jusqu'à 1Khz pour des objets de complexité moyenne (jusqu'à une centaine de points matériels et d'éléments de liaison). Il sera ainsi envisageable de faire des traitements en temps différé sur des objets plus complexes dans des conditions de temps raisonnables. Le temps d'exécution doit être fixe dans la mesure du possible, pour permettre
- d'évaluer les facteurs de suréchantillonnage ,
- de faire la synchronisation sur l'horloge commune.

b) optimisation sur l'occupation mémoire programme et mémoire données. Ce facteur répond en partie à un accroissement rapide de complexité des objets à simuler entraînant une demande croissante de mémoire. Ce facteur doit faire par ailleurs face au coût encore relativement élevé des éléments mémoire utilisés.

c) rapidité de génération du contexte au niveau du processeur hôte et du chargement de ce contexte (ou une partie) dans le processeur de simulation. Les traitements relatifs à cette génération doivent prendre en considération la limitation mémoire sur le processeur hôte et du temps de réponse à l'utilisateur qui fait une demande d'expérimentation.

d) rendre opératoire une génération "dynamique" de contexte pour

répondre à la notion de modification de structure. Cette notion regroupe deux aspects :

- un premier aspect recouvre une modification "temps réel" de la structure expérimentable permettant de percevoir "en continu" sur l'image l'adjonction ou à la suppression de nouveaux éléments. Cette situation correspond à l'adjonction ou la suppression de modules de simulation (au niveau code et données) à partir du contexte courant. Nous faisons abstraction ici des problèmes de cohérence énergétique occasionés par cette modification. Cet aspect n'a pas fait l'objet d'une expérimentation même si la structuration du contexte adopté le prend en considération. Ce type de modification soulève des problèmes encore latents qui doivent être pris en compte par le processus dialogue. Ceci concerne notamment le "marquage" des éléments à prendre en compte pour l'adjonction ou la suppression.

- un deuxième aspect concerne la modification "dynamique" où la simulation s'exécute avec l'ensemble de variables initiales relatives à une structure ayant subi une modification de façon interactive par l'opérateur. Cette situation implique donc une coupure momentanée de l'image. Nous incluons dans ce type de modification le cas d'un changement d'objet à expérimenter (suite à un passage sur la classe de commande INIT).

6.2. Enoncé du problème

Nous pensons que les objectifs cités mettent en jeu des demandes contradictoires (ex. rapidité de génération d'un contexte et rapidité d'exécution du code) et il est nécessaire de trouver un compromis. L'utilisation du matériel actuel a facilité en partie notre travail notamment dans l 'évaluation des temps de calcul, du fait d'avoir un temps fixe d'exécution de l'instruction et d'avoir des traitements vectorisés.

Nous supposons dans cette étude que nous disposons de l'ensemble de procédures relatives à la résolution des équations régissant chaque module de simulation, une proposition sur la réalisation de ces procédures ayant fait l'objet paragraphe (5), il s'agit ici :

- d'une part d'étudier la génération d'un code relatif à une structure de programme permettant pour une structure expérimentable de faire l'appel à l'ensemble des procédures adéquats et permettant de retrouver les adresses des données manipulées par chaque procédure appelée,
- d'autre part d'étudier la structure des données manipulées par les procédures.

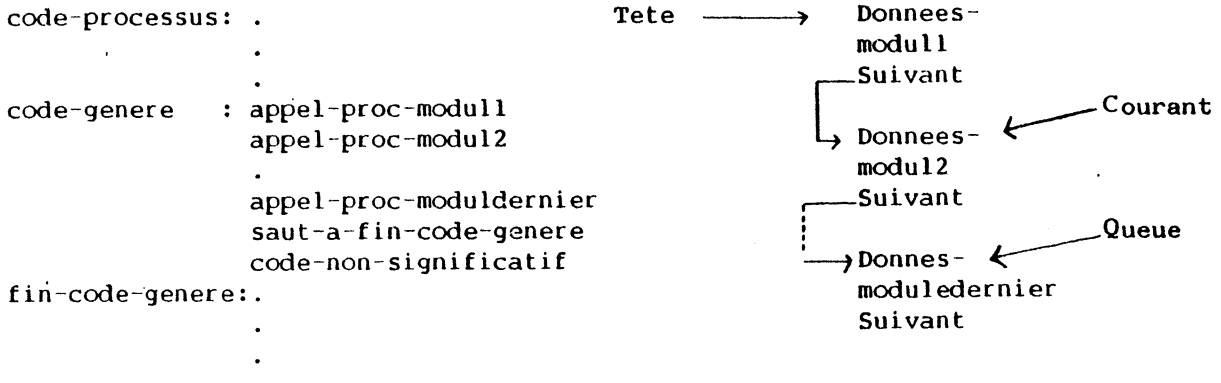
En effet le modèle de simulation que nous avons adopté ne nécessite

pas la génération dynamique du code des procédures, en raison de l'isomorphie entre les modules de simulation créés et les résolutions individuelles des équations qui les représentent. Il est donc possible de constituer statiquement les procédures et intégrer de manière dynamique le code des appels dans le code du processus de simulation.

6.3. Structuration du code du processus

6.3.1. Etude de cas

Pour orienter notre choix, nous exposons un certain nombre de structurations possibles en partant des cas défavorables et en mentionnant pour chacune les inconvénients immédiats par rapport aux objectifs cherchés. Pour nous un premier schéma évident d'une structuration de code qui satisfait l'isomorphie entre les modules existants et les procédures à exécuter est le suivant :



Ceci suppose que le code relatif à la séquence d'appels, les instruction de saut, les blocs de données sont générés de manière dynamique. Un bloc de données contient l'ensemble d'informations nécessaire à la procédure pour s'exécuter. Une étude détaillée des données est faite pour la structuration de code choisie.
La variable pointeur Courant permet de retrouver le bloc de données pour la procédure couramment exécutée. La fonction de la procédure est alors complété par la mise à jour de la variable Courant en faisant l'affectation :
Courant <- Suivant
avant chaque retour de procédure.

Les variables pointeurs Tete et Queue permettent la modification et le parcours des listes dans la phase de génération de données. Pour la génération du code d'appel la différence entre les deux adresses

absolues (adresse du code d'appel courant, adresse de la procédure appelée) permet de calculer la valeur du champ déplacement.

Le chaînage sur les blocs de données permet :
- une simplification en cours d'exécution dans la recherche des adresses des donneés. Celle ci se fait à partir d'un adressage relatif à la valeur du pointeur courant,
- une facilité de gestion de la mémoire libre pour une modification qui revient à faire une mise à jour de chaînage des blocs de données.
Au niveau de la modification du code, une suppression ou adjonction peut nécessiter une translation du code des appels après celui supprimé ou ajouté. Nous supposons en fait qu'un ordre d'exécution peut être nécessaire pour respecter une cohérence du modèle mécanique et n'implique pas forcément une insertion en queue.
La deuxième illustration de ce fonctionnement est celle qui permet d'éviter les translations en faisant succéder la génération d'un appel par une instruction de saut inconditionnel à l'instruction d'appel suivant (ou à l'étiquette fin-code-généré si c'est le dernier appel). Les modifications dans ce cas reviennent à faire des mises à jour du champ déplacement dans les instructions de saut et nécessite une gestion de la mémoire programme libre.

6.3.2. Critiques

La première architecture que nous avons donné a été simulé sur le processeur LSI. Nous pensons :

- qu'elle représente une amélioration importante par rapport à celle qui a été adoptée dans le premier programme d'expérimentation (ANIMA-I 81) [4.]. Dans celui ci on fait appel statiquement à des macros-instructions et on ne permet pas à l'opérateur de faire une modification interactive de l'objet,
- qu'elle nécessite la réalisation du programme avec un langage d'assemblage permettant la manipulation des adresses absolues.

- qu'elle n'est pas très adaptée au processeur vectoriel pour les raisons suivantes :
  . les ruptures de séquence occasionnées par instructions de saut et les appels aux procédures de simulation ne favorisent pas un traitement vectorisé,
  . le traitement de la procédure de simulation elle même dans sa définition actuelle ne profite pas à un traitement vectorisé. En effet les équations des modules met en jeu des expressions de type somme (Ai*Bi) avec un i petit (jusqu'à 3) et ne nécessite pas l'utilisation d'un programme sous forme de boucle "repliée" sur les indices i, ni sous forme d'appels à des procédures de bibliothèque,

. la gestion des pointeurs est lourde en raison de l'absence d'instruction d'indirection, d'instruction d'adressage indexé et en raison des temps d'accès successifs à la mémoire centrale. Les instructions non élémentaires de calcul d'adresse sont en effet à éviter.

6.3.3. Solution proposée

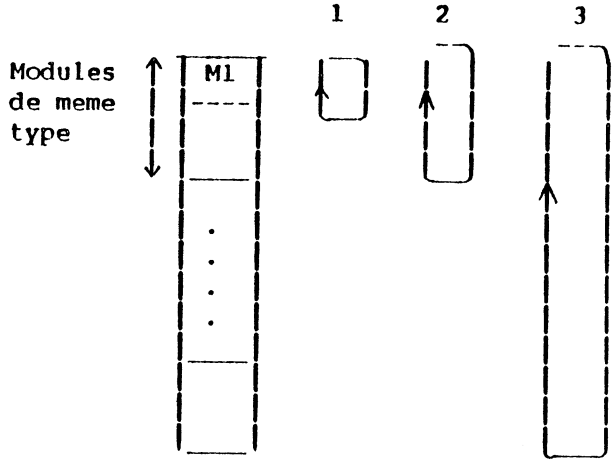
Une structuration plus adaptée est proposée ici, elle présente les modifications suivantes par rapport à la deuxième illustration citée :

- utilisation de procédures qui traitent chacune un ensemble de modules de même type c'est à dire régis par le même algorithme, cela réduit le nombre d'appels,
- organisation des données sous forme de vecteurs dont les adresses de début sont connus au moment de l'exécution des procédures. Les données attribuées à chaque procédure sont représentées par un ou plusieurs vecteurs de même longueur. Nous appelons cette longueur le "facteur de multiplicité" correspondant à la multiplicité des modules traités par la procédure. Ce facteur est utilisé par la procédure comme compteur dans le traitement itératif.
- utilisation d'un code exécutable indépendant de l'objet à expérimenter. Cela consiste à faire les appels systématiques à toutes les procédures en adoptant un facteur de multiplicité nul pour les vecteurs fictifs associés aux modules inexistants.

Celle-ci introduit des simplifications importantes car elle permet :

. d'avoir une définition statique du code,
. de supprimer les instructions de saut,
. de nous faciliter le travail en évitant de faire une génération dynamique du code sur le processeur vectoriel qui aurait été fastidieux, et limiter ainsi le temps de préparation du contexte pour le processus.

D'une façon générale les possibilités de vectorisation peuvent être décrites par le schéma suivant :



Le dernier type de vectorisation n'a pas été mis en oeuvre, dans la mesure où les facteurs suivants ne permettent pas de l'exploiter facilement :
- multiplicité différente possible pour les différents types de module, posant des problèmes pour le contrôle de boucle,
- contraintes sur l'ordre d'exécution des algorithmes relatifs à des types de modules différents, à cause des dépendances de variables. Cette dépendance lie notamment les variables pour les modules de type matériel et celles pour les modules de type liaison.

Nous illustrons ce dernier problème par l'exécution en parallèle (parallélisme interne sur les instructions) des 2 traitements cycliques suivants :
1- calcul des positions d'éléments matériels : Pi
2- calcul des forces de liaison Fj= f(Pu, Pv)
L'exécution ne garantie pas ici le calcul d'une force de liaison à partir des positions matériels issues d'un même cycle.

6.3.4. Structuration des données

Il s'agit ici de la description de l'ensemble des données figurant dans le contexte du processus de simulation. Cet ensemble regroupe :

a) l'ensemble des vecteurs mis en jeu dans l'exécution vectorisée des procédures. Nous différencions à ce titre les vecteurs suivants :

1- vecteurs de variables de force et de position représentant l'état d'équilibre des modules matériels et des modules de liaison dans le système mécanique,

2- vecteurs de paramètres mécanique pour l'univers de simulation,
3- vecteurs de variables représentant les arguments définis dans la représentation interne de l'objet. Ces variables sont constituées de les trois types correspondant aux trois types suivants :

. pour un module de liaison les adresses des variables positions des éléments matériels représentant les attaches.

. pour un module d'entrée les adresses des entrées adéquats dans les tableaux contenant l'échantillon de geste et l'échantillon de retour d'effort.

. pour un module contrôle de paramètre les adresses de paramètres mécaniques pour un module de contrôle de paramètres.

b) le tableau contenant l'échantillon gestuel courant et le tableau contenant l'échantillon de retour d'effort courant. Ces tableaux permettent la communication en DMA des échantillons avec l'interface gestuelle,
c) les 2 tableaux contenant les 2 derniers échantillons visuels avant le démarrage du processus de visualisation. Ces tableaux sont communs à ce dernier et au processus de simulation. Il contient dans le cas de l'affichage temps réel la succession de commandes pour le générateur de vecteur.
d) les vecteurs de travail nécessaire au calcul des variables de positions et de force. Ils constituent notamment des variables intermédiaires dans la décomposition des équations en équations élémentaires dont la résolution utilise les procédures de bibliothèque.
e) le tableau d'adresses de référence pour chaque type de module et le tableau de facteurs de multiplicité.
f) l'ensemble des variables suivantes :

. drapeaux de débordements,
. drapeaux de synchronisation,
. facteurs d' échantillonnage.

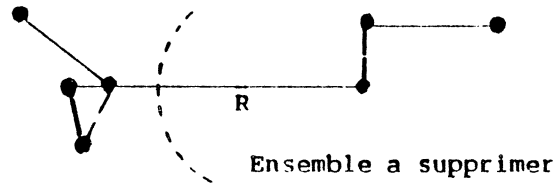
6.4. Modification temps réel d'objet

Nous pensons que cette modification est envisageable si sa prise en compte permet de :
- garder les conditions courantes de fonctionnement du processus de simulation (utilisation de surchantillonnage ou sous-échantillonnage, synchronisation entre les processus),
- minimiser les perturbations sur l'affichage en essayant de garder la linéarité sur les images,
- garder d'une part la cohérence au niveau de la structure des données et d'autre part la cohérence au niveau des états des variables représentant l'équilibre du système mécanique. Le traitement de ce dernier point nécessite donc une exclusion

mutuelle entre une modification du contexte par le processus dialogue et l'exécution du processus de simulation.

Compte tenu des deux premiers points il s'agit surtout de trouver un traitement efficace qui remplacerait par exemple la mise à jour rapide des données à partir du chaînage entre les blocs (évoquée dans la première structuration des données). Nous pensons qu'une opération comme la translation sur les vecteurs n'est pas à envisager.

La solution proposée ici est l'adoption d'un facteur de multiplicité maximum par type de module lors d'un premier chargement global du contexte, permettant de fixer une borne supérieure sur le temps d'exécution du processus. Les cas de modifications et leur traitement sont alors les suivants :
1) suppression de modules matériels
Traitement : vide
2) suppression de modules de liaison
Traitement : mise à zéro des paramètres mécaniques. Ceci entraîne l'annulation des forces appliquées sur les éléments matériel attaches. La mise a ' zéro systématique n'est pas optimum comme le montre la figure où il suffit de détruire une partie de l' ensemble des liaisons (ici constitué par le ressort R). Cette optimisation n'est pas abordée ici.



3) suppression de modules de visualisation
Traitement : attribution d'une ou de deux adresses (suivant un module point ou un module vecteur) d'une variable position représentant une attache fictive située en dehors de la fenêtre de visualisation. Cette variable fait partie des variables de travail.
4) création de module
Traitement: mise à jour des paramètres mécaniques et des variables entrant dans l'exécution du module (variales forces, positions, arguments).

La solution qui a été évoquée ne prévoit pas de modification au niveau du code du processus. Chaque traitement utilise un transfert ou plusieurs transferts de données entre le processeur hôte et le processeur vectoriel. Nous décrivons par la

succession des points suivants un algorithme permettant d'effectuer ces transferts en exclusion mutuelle avec l'exécution du processus :

. préparation des blocs de données à transférer,
. blocage de l'horloge de synchronisation des processus,
. attente de la terminaison du processus de simulation pendant un temps maximum égal à la période d'horloge de synchronisation,
. transfert des blocs par DMA,
. déblocage de l'horloge.

Remarques

La troisième fonction (attente) évite d'introduire pour ce processus une instruction supplémentaire signalant cette terminaison (mise à jour d'un drapeau ou interruption du hôte). L'algorithme respecte la cohérence au niveau de l'état des variables du système mécanique mais il introduit une variation au niveau du temps d'intégration des équations, temps qui est majoré par : période d'horloge + temps de transfert des blocs. L'intervalle de temps séparant l'affichage de deux échantillons visuels est aussi majoré par ce temps.

6.5. Conséquences sur la modification interactive

Nous disons à priori que cette modification recouvre les deux formes suivantes :

. une modification locale nécessitant la préparation de blocs de données et leur transfert dans le processeur vectoriel relatifs à la suppression ou à l'ajout de modules. Ces opérations représentent une partie du traitement de la modification temps réel,
. une modification globale nécessitant la réinitialisation des variables positions et variables forces pour les modules restants. Le cas de translation sur des éléments matériels est alors traité par celle-ci.

Nous pensons qu'une génération du contexte "quasi-instantanée" n'est pas indispensable dans la mesure où l'utilisateur renouvelle une expérimentation et ne cherche pas une continuité par rapport à l'affichage courant. Nous assimilons par conséquent le traitement de la modification interactive à celui de la génération du contexte global (utilisé dans une première expérimentation de structure expérimentable), génération qui prend actuellement un temps maximal de 2 secondes. Celle-ci est assurée par une fonction du processus dialogue décrite plus loin

que nous avons appelé Interface LSI-AP pour la génération globale de contexte.

6.6. Problèmes latents pour la modification temps-réel et pour la modification interactive

La génération globale d'un contexte se fait à la suite de la commande de demande d'expérimentation, présentée sur le menu de la classe racine. Cette génération utilise la représentation interne de la structure courante définie par l'ensemble des listes linéaires. Une modification temps réel ou une modification interactive utilisant une modification locale pose les problèmes suivants :
- gestion d'un "marquage" au niveau de la représentation interne permettant de référencer les variables à mettre à jour au niveau du contexte courant du processus de simulation.
- la représentation courante des vecteurs au niveau de l'hôte permettant le calcul des adresses pour le transfert des variables précédentes,
- la différenciation de ces deux types de modification qui doit se traduire par l'introducion de plusieurs modes utilisateur auxquels seront associés la commande d'expérimentation.

Nous parlerons essentiellement par la suite de la génération globale du contexte dans la mesure où celui satisfait de façon raisonnable au niveau du temps de réponse la modification interactive. Nous pensons que l'étude sur la modification temps réel que nous avons faite en terme de procédures à implanter sur le hôte consiste en un premier débrousaillage.

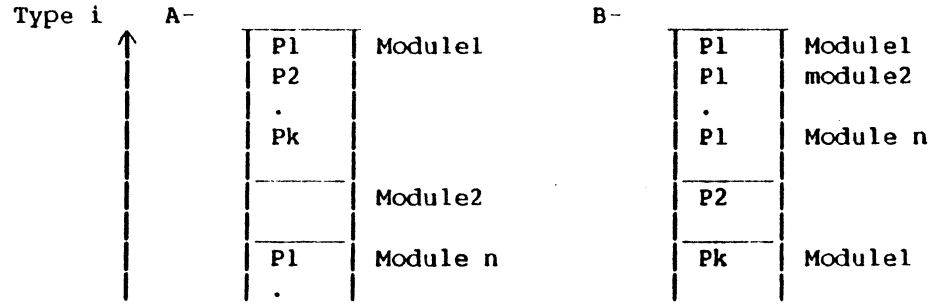
Nous pensons aussi que les problèmes évoqués ci-dessus ne sont pas des problèmes complexes, néanmoins l'introduction de nouveaux modes utilisateur se traduit à priori par une complexité supplémentaire au niveau du langage de commandes que nous avons mis de côté dans cette première réalisation.

6.7. Optimisation de la place mémoire

Pour le traitement vectorisé qu'elle met en oeuvre, chaque procédure de simulation nécessite :
- la connaissance de l'adresse du premier élément d'un vecteur pris comme référence,
- la connaissance des déplacements relatifs par rapport à cette adresse des vecteurs restants,
- la connaissance du facteur de multiplicité commune à l'ensemble des vecteurs.

Par souci de rapidité d'exécution il aurait été profitable de définir des adresses absolues de référence apparaissant au niveau du code programme. Ceci aurait impliqué une répartition statique des bancs de mémoire de données. En raison des multiplicités différentes possibles par type de module nous pensons que cette répartition n'est pas adaptée et nous avons choisi une répartition variable suivant les facteurs de multiplicité définis par l'utilisateur. Cela nécessite une première lecture d'une adresse de référence et une lecture du facteur de multiplicité à chaque appel de procédure nous amènant à introduire le tableau des adresses de référence et le tableau des facteurs de multiplicité dans la structure de données. Ces tableaux sont placés au niveau de la mémoire MD dans la mesure où les bancs de registres rapides SPAD et DATAPAD servent de registre de travail et se prêtent difficilement à une mémorisation de données communes à plusieurs procédures.

Nous considérons ici deux organisations simples des vecteurs qui facilite de manière évidente le calcul des adresses de référence.



Nous pouvons choisir ici un ordre arbitraire dans le remplissage de la mémoire. L'adresse de référence pour la procédure de simulation de rang i est calculée par :
ADRi = sigma NBVECTj * FMj + A0 pour j:=1 à i-1
où NBVECTj est le nombre de vecteurs mis en oeuvre pour le type de module de rang j dans le remplissage de la mémoire, FM le facteur de multiplicité et A0 l'adresse de début de remplissage. Pour une génération de contexte donnée nous prenons par défaut les FMk relatifs à la précédente génération.

6.8. Interface AP-LSI pour la génération de contexte

Cette interface prend en charge les deux étapes suivantes :
1- transformation des données de la représentation interne à un ensemble de données intermédiaires,
2- transfert des données intermédiaires grâce aux fonctions de l'interface matérielle (formattage des données, transfert

programmé, transfert DMA) permettant la génération des vecteurs.

La première opération consiste à faire :
. les transformations d'unités au niveau des paramètres mécaniques et géométriques,
. le calcul des variables adresses.
. le remplissage du tampon des données intermédiaires.

La deuxième opération consiste à gérer le transfert du tampon par blocs d'éléments contigus de même format, imposé par le transfert DMA. La structure du tampon et la décomposition en blocs sont directement liés à l'organisation des vecteurs situés au niveau du contexte du processus de simulation. Une étude comparative des deux organisations des vecteurs introduits au paragraphe précédent est faite ici :

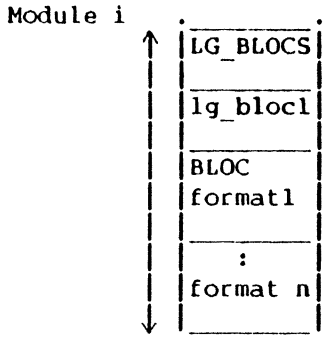
1- dans la première configuration il s'agit de rendre contigOs les éléments de même rang dans les différents vecteurs, permettant de grouper en mémoire contigue les données pour un module donné. Cela permet de :
- régler la taille du tampon suivant la quantité mémoire disponible suivant que le transfert s'effectue par module, sur l'ensemble des modules d'un même type ou sur la totalité des modules.
- faciliter le remplissage du tampon par l'ensemble des données intermédiaires relatives à un module. Ces dernières sont en effet calculées à partir des données contenues dans l'élément de liste courant lors d'un parcours des listes. Les données relatives à un module d'entrée nécessite en complément le parcours de la liste des transducteurs et de la liste des canaux.

2- dans la deuxième configuration, on a un déplacement d'une unité entre les éléments des vecteurs. La constitution d'un vecteur d'éléments contigOs nécessite le parcours de l'ensemble des listes. Il est par conséquent intéressant de faire la constitution du tampon pour la totalité des modules simultanément au parcours des listes.

Choix de la configuration

Nous avons adopté la première configuration pour les raisons suivantes :
. elle a permis une réservation mémoire minimum sur le processeur hôte qui s'est traduit par la constitution d'un tampon pour un transfert par type de module.
. elle permet une modification rapide du contexte dans le cas d'une modification temps réel,

. elle permet de connaître statiquement au moment de l'implantation des procédures de simulation les déplacements entre les éléments d'un vecteur et les déplacements entre vecteurs. Cela n'est pas le cas pour la deuxième configuration où le déplacement entre les vecteurs est fonction du facteur de multiplicité.
Cette configuration présente l'inconvénient suivant : les données intermédaires peuvent présenter des formats variables (entier ou réel) et il a été impératif de regrouper les données de même format pour minimiser les transferts de base.
Nous décrivons ici la structuration du tampon adopté :



Le schéma général de l'interface LSI-AP pour la génération du contexte global est alor le suivant :

```
Schéma interface_LSI_AP

blocage_horloge_commune;
attente_terminaison_simulation;
Pour tout type_de_module faire
     constituer_tampon(type_de_module);
     calcul_adresse_reference(type_de_module, adresse_AP);
     mise_a_jour_adr_DMA(adress_AP);
     premier(bloc_de_transfert);
     Tant que non dernier_bloc faire
          transfert_DMA(zone_bloc, lg_bloc);
          suivant(bloc_de_transfert);
     Fintantque
Finpour
deblocage_horloge;

Finschema
```

Remarques

La procédure ci-dessus fait une attente de la terminaison du processus de simulation au lieu de commander l'arrêt de ce

processus par une commande d'arrêt du processeur. Ceci permet au processus d'entretien de l'image de s'exécuter sur la dernière image affichée pendant la phase de transfert des données.
Les primitives premier, suivant et dernier sont les primitives "classiques" d'avancement et de test dans la file séquentielle des blocs du tampon.
L'adresse adresse_AP est calculée une fois à chaque nouveau type à traiter; elle est mise à jour automatiquement au niveau de l'interface matérielle qui gère le DMA pour les transferts successifs des zones.

CONCLUSION

Nous pensons que la maquette réalisée actuellement a permis :
- la validation du modèle choisi relative en particulier à l'introduction des modules 2D et des modules autres que la masse ponctuelle et la liaison simple ressort-frottement,
- une évaluation rigoureuse du modèle. Celle-ci a été facilitée par les possibilités de contrôle fournies à l'utilisateur et par une souplesse dans la modification des objets,
- de montrer la viabilité du système dans le cadre d'une expérimentation multisensorielle "pertinente".

Ce travail a donné l'occasion de nous confronter à nombre de problèmes relèvant de la construction d'un système interactif et aux exigeances de diverses situations "temps-réel". Nous avons pu tirer les constatations générales suivantes dans le but de faire évoluer le système :
- au niveau de la construction des objets : les commandes de l'utilisateur conçues actuellement agissent à "bas-niveau" c'est à dire manipulent essentiellement des éléments de base. Des niveaux d'abstraction comme ceux mentionnés au paragraphe (II 2.4.4.c) nous semblent indispensables.
- au niveau de la synchronisation des processus pour le fonctionnement temps réel : nous pensons que le mécanisme adopté a été rendu possible par une évaluation précise des temps d'exécution et de l'existance d'une moyenne commune à divers temps d'exécution. Il nous semble qu'il est indispensable d'étudier un modèle de synchronisation plus général où les seules contraintes sont à priori dictées par les hypothèses suivantes :
  . nécessité d'un échantillonage constant des signaux gestuels,
  . nécessité d'un échantillon constant des résultats de la simulation.

Nous pensons que ce type de contraintes n'impose pas forcement des vitesses fixes aux divers processus dans leur décomposition interne (ex. existance de plusieurs simulateurs communicants entre eux).

Pour une diffusion des résultats à court terme nous supposons que :
- une période d'apprentissage est nécessaire pour appréhender des ensembles de valeurs de paramètres conformes à des comportements cherchés. Nous avons abordé cette phase par la construction de bibliothèques d'objets simples.
- les fonctions réalisées permettent de produire et

d'expérimenter des "évènements sensoriels". Ces fonctions doivent être complétées d'une part par la possibilité de faire des modifications "temps-réel" et d'autre part par l'exploitation de ressources nécessaires à une phase compositionnelle.
- les problèmes de capacité mémoire existant sur le hôte peuvent être surmontées sans remettre en cause l'architecture du système par le transport des programmes permettant de constituer le code du processus dialogue, sur une machine récemment acquise (VAX 730). Les deux machines sont actuellement reliées par une interface parallèle qui disposent de facilité d'interruption du côté LSI 11/02. Cette interface permettra essentiellement à celui-ci de recevoir les commandes et les données nécessaire au chargement du contexte de simulation.

En ce qui concerne les possibilités d'aboutir à un outil commun apte à fournir des ressources pour une expérimentation sonore notons que :
- le système actuel a permis l'intégration de modules de base (réalisés par J.L. Florens) moyennant l'utilisation d'un facteur de suréchantillonnage élevé lors de leur simulation (transduction accoustique des valeurs mécaniques à 15KHZ). Le problème essentiel à ce niveau est la coexistance de 2 facteurs de suréchantillonnages : la première utilisée pour la simulation des modules "lents" utilisés pour la production visuelle, dont les variables forces ont servi à exciter les variables positions des modules "rapides" simulés à avec un 2eme facteur de suréchantillonnage plus élevé,
- les outils actuels de dialogue, les principes de décomposition en processus et leur synchronisation nous ont permis de réaliser des programmes qui mettent en oeuvre des objets "types", de structure figée mais permettant la modification de l'ensemble des paramètres. Dans ce cas les modules mis en jeu ont été simulés avec le même facteur de suréchantillonnage élevé et ont donné des résultats très satisfaisiants pour une production simultanée visuelle et sonore [48].

ANNEXE

ANNEXE 1

Nous présentons ici quelques exemples types de programmation des modules de simulation : boucle unique optimisée (MASSE.APA), boucle unique optimisée et repliée (MASS1.APA), appel à des procédures mettant en oeuvre des boucles traitant des fonctions élémentaires (LICX.APA).

```
        " MASSE.APA
        " PROCEDURE DE SIMULATION MASSE
        " X(N-2)=X(N-1)
        " X(N-1)=X(N)
        " X(N)=1/M * FX(N-1) +2 * X(N-1) - X(N-2)
        " FX=0
        "
        " LE MODULE VMAS REALISE LA FONCTION CI-DESSUS POUR LES VECTEURS DE
        " ET CELUI DES Y, RESPECTIVEMENT AUX 2 APPELS CONSECUTIFS DE LA BOU
        "
        $TITLE MASSE
        $ENTRY VMAS
        "
        "SPADS UTILISES
        INC2=0
        INC7XY=1
        INCXY=2
        DEC4=3
        ADVMAS=4
        CPTMAS=5
        DEC3XY=6
"       TRAV: 14,15
"
"
VMAS:   DB=2; LDSPI INC2
        DB=3; LDSPI DEC3XY
        DB=7; LDSPI INC7XY
        DB=4; LDSPI DEC4
        LDMA;DB=17666                   " AD VECT. MASSES
        DB=13; LDSPI INCXY
        LDMA;DB=17665                   " AD NB MASSES
        DB=MD; LDSPI 14
        MOV 14,ADVMAS
        DB=MD;LDSPI 15
        ADD DEC3XY,ADVMAS
        MOV 15,CPTMAS
        SUB INCXY,ADVMAS
        JSR BCLMAS                      " VEC X
        MOV 14,ADVMAS
        ADD DEC4,ADVMAS
```

```
        DB=4;LDSPI DEC3XY
        DB=10; LDSPI INC7XY
        SUB INCXY,ADVMAS
        MOV 15,CPTMAS
        JSR BCLMAS                          " VEC Y
        RETURN
"
BCLMAS: ADD INCXY,ADVMAS;SETMA              " X(N-1)
        NOP
        SUB INC2,ADVMAS;SETMA               " X(N)
        NOP
        ADD DEC4,ADVMAS;SETMA;MI<DB;DB=MD       " X(N-2)=X(N-1)
        NOP
        SUB INC2,ADVMAS;SETMA;MI<DB;DB=MD       " X(N-1)=X(N)
        NOP
        FADD ZERO,ZERO ; SUB DEC3XY,ADVMAS;SETMA        "1/M
        FADD
        ADD INC7XY,ADVMAS;SETMA                 " F(N-1)
        DPX(0)<MD                               " DPX=1/M
        MOV ADVMAS,ADVMAS; SETMA; MI<DB; DB=ZERO" INIT F(N-1)
        FMUL DPX(0),MD                          " 1/M * F(N-1)
        FMUL;   SUB DEC4,ADVMAS; SETMA          " X(N-1)
        FMUL
        FADD FM,FA;     RPSF K2; DPX(0)<DB      " 2. ,SOM 1
        FADD; FMUL DPX(0),MD                    " 2. * X(N-1)
        FMUL
        FMUL;ADD INC2,ADVMAS; SETMA             " X(N-2)
        FADD FM,FA                              " SOM 2
        FADD
        DPX(0)<MD
        FSUBR DPX(0),FA                         " SOM 3 ( FA COUR - X N-2 )
        FSUBR; DEC CPTMAS
        MI<FA; SUB DEC4,ADVMAS; SETMA;BGT SUIT          "  X(N), RESULT X(N
        RETURN
SUIT:   JMP BCLMAS
K2:     $FP 2.
        $END
```

Le programme suivant fait intervenir un "repliement" de la boucle optimisée figurant dans le programme précédent. Cela réduit le temps d'exécution d'environ de moitié. Quelques simples modifications ont été introduites par rapport à la boucle initiale conséquentes à des sauvegardes intermédiaires de données (dans des SPADs et DATAPADs) pour assurer une indépendance sur l'exécution des deux blocs.

```
        " MASS1.APA
        "
        $TITLE MASS1
        $ENTRY VMAS
        "
        INC2=0
        INC7XY=1
        INCXY=2
        DEC4=3
        ADVMAS=4
        CPTMAS=5
        DEC3XY=6
        SAUV1 =10
        SAUV2 =12
"       TRAV: 14,15
"
"
VMAS:   DB=2; LDSPI INC2
        DB=3; LDSPI DEC3XY
        DB=7; LDSPI INC7XY
        DB=4; LDSPI DEC4
        LDMA;DB=17666                   " AD VECT. MASSES
        DB=14; LDSPI INCXY
        LDMA;DB=17665                   " AD NB MASSES
        DB=MD; LDSPI 14
        MOV 14,ADVMAS
        DB=MD;LDSPI 15
        ADD DEC3XY,ADVMAS
        MOV 15,CPTMAS
        SUB INCXY,ADVMAS
        JSR DEBMAS                      " VEC X
        MOV 14,ADVMAS
        ADD DEC4,ADVMAS
        DB=4;LDSPI DEC3XY
        DB=10; LDSPI INC7XY
        SUB INCXY,ADVMAS
        MOV 15,CPTMAS
        JSR DEBMAS                      " VEC Y
        RETURN
```

```
"
DEBMAS: ADD INCXY,ADVMAS;SETMA          " X(N-1)
        NOP
        SUB INC2,ADVMAS;SETMA           " X(N)
        NOP
        ADD DEC4,ADVMAS;SETMA;MI<DB;DB=MD       " X(N-2)=X(N-1)
        NOP
        SUB INC2,ADVMAS;SETMA;MI<DB;DB=MD       " X(N-1)=X(N)
        NOP
        FADD ZERO,ZERO ; SUB DEC3XY,ADVMAS;SETMA        "1/M
        FADD;MOV ADVMAS,SAUV1
        ADD INC7XY,SAUV1;SETMA;DPX(3)<FA                        " F(N-1)
        DPX(0)<MD                               " DPX=1/M
        MOV SAUV1,SAUV1; SETMA; MI<DB; DB=ZERO" INIT F(N-1)
        FMUL DPX(0),MD                          " 1/M * F(N-1)
        FMUL;   SUB DEC4,SAUV1; SETMA           " X(N-1)
BCLMAS: ADD INCXY,ADVMAS;SETMA;                 FMUL
        NOP;                                    FADD FM,DPX(3);RPSF K2;DPX(
        SUB INC2,ADVMAS;SETMA;                  FADD; FMUL DPX(1),MD
        NOP;                                    FMUL;MOV SAUV1,SAUV2
        ADD DEC4,ADVMAS;SETMA;MI<DB;DB=MD;      NOP
        NOP;                                    FMUL;ADD INC2,SAUV2;SETMA
        SUB INC2,ADVMAS;SETMA;MI<DB;DB=MD;      NOP
        NOP;                                    FADD FM,FA
        FADD ZERO,ZERO;SUB DEC3XY,ADVMAS;SETMA; DPX(2)<MD
        FADD;MOV ADVMAS,SAUV1;                  FSUBR DPX(2),FA
        ADD INC7XY,SAUV1;SETMA;DPX(3)<FA        FSUBR
        DPX(0)<MD;                              MI<FA;SUB DEC4,SAUV2;SETMA
        MOV SAUV1,SAUV1;SETMA;MI<DB; DB=ZERO;   NOP
        FMUL DPX(0),MD;                         DEC CPTMAS
        FMUL;   SUB DEC4,SAUV1; SETMA;          BGT BCLMAS
        RETURN
K2:     $FP 2.
        $END
```

Le programme suivant momtre un exemple de décomposition de la résolution d'une équation en opérations simples qui permet l'appel à des procédures externes et des procédures déjà existantes dans la bibliothèque.

```
" LICX.APA
"
" MODULE DE SIMULATION PSEUDO-LIAISONS CONDITIONNELLES.
" LA LIAISON S'EFFECTUE APRES CONDITION SUR LES X DES ELEMENTS MATERIELS.
" GENERATION D'UNE LIAISON A 1 DEGRE (SUR X) SI CONDITIONS REMPLIES.
"
"
"       $TITLE LICX
        $ENTRY LICX
        $EXT VMUL,CVSUB,CVNEG,VADD,VSUB   " PROC. BIBLIOTHEQUE
        $EXT SETMOV,SOMFXY                " PROC. EXTERNES
"
" SETPADS UTILISES
"
        ADSEUIL=0
        SEPT=0
        ADF=2
        QUATRE=3
        TRAV=12
        ADF2=15
        INCFY=16
        DRFY=13
        RFY=14
        ADAT=1
        ADTR=4
        NLIC=5
        INCY12=6
"
LICX:   LDSPI ADF2;DB=2          " GENER VECTEUR Y2N
        LDMA;DB=17646
        LDSPI ADTR;DB=14774              " TRAV= 15000
        LDMA;DB=17645
        LDSPI RFY;DB=MD
        LDSPI SEPT,DB=7
        LDSPI DRFY;DB=MD
        LDSPI INCY12;DB=0
        MOV RFY,ADAT
        ADD ADF2,ADAT
        MOV DRFY,NLIC
        LDSPI QUATRE,DB=4
        JSR SETMOV
        MOV RFY,ADAT                     " GEN Y2N-1
        ADD ADF2,ADAT
```

```
        MOV DRFY,NLIC
        LDSPI ADTR;DB=14775             " TRAV= 15000
        LDSPI INCY12;DB=2
        JSR SETMOV
        MOV RFY,ADAT                    " GEN Y1N
        MOV DRFY,NLIC
        LDSPI ADTR;DB=14776             " TRAV= 15000
        LDSPI INCY12;DB=0
        JSR SETMOV
        MOV RFY,ADAT                    " GEN Y1N-1
        MOV DRFY,NLIC
        LDSPI ADTR;DB=14777             " TRAV= 15000
        LDSPI INCY12;DB=2
        JSR SETMOV
Y2NMY1: LDSPI 0;DB=15000                " GENER Y2N-Y1N, Y2(N-1)-Y1(N-1)
        LDSPI 1;DB=4
        LDSPI 2;DB=15002
        LDSPI 3;DB=4
        LDSPI 4;DB=15000
        LDSPI 5; DB=4
        MOV DRFY,6
        JSR CVSUB
V2MV1:  LDSPI 0;DB=15001                " Y2N-Y1N-(Y2.N-1.-Y1.N-1.)
        LDSPI 1;DB=4
        LDSPI 2;DB=15000
        LDSPI 3;DB=4
        LDSPI 4;DB=16000                " TRAV =16000
        LDSPI 5;DB=2
        MOV DRFY,6
        JSR VSUB
VCOND:  MOV RFY,0               " Y2N-Y1N-SEUIL
        LDSPI 1;DB=7
        ADD 1,0
        DEC 0
        MOV 0,12
        LDSPI 1;DB=7
        LDSPI 2;DB=15000
        LDSPI 3;DB=4
        LDSPI 4;DB=15000
        LDSPI 5;DB=4
        MOV DRFY,6
        JSR VSUB
VFZ:    LDSPI 0;DB=16000                " Z*(V2-V1)
        LDSPI 1;DB=2
        DEC 12
        MOV 12,2
        LDSPI 3;DB=7
        LDSPI 4;DB=16000
```

```
        LDSPI 5;DB=2
        MOV DRFY,6
        JSR VMUL
VFK:    LDSPI 0;DB=15000              " K(Y2-Y1-SEUIL)
        LDSPI 1;DB=4
        DEC 12
        MOV 12,2
        LDSPI 3;DB=7
        LDSPI 4;DB=16001
        LDSPI 5;DB=2
        MOV DRFY,6
        JSR VMUL
VSOMF:  LDSPI 0;DB=16000              " F=FK+FZ
        LDSPI 1;DB=2
        LDSPI 2;DB=16001
        LDSPI 3;DB=2
        LDSPI 4;DB=16000
        LDSPI 5;DB=1
        MOV DRFY,6
        JSR VADD
"
TSTLIC: LDSPI ADSEUIL,DB=14774
        LDSPI QUATRE;DB=4
        LDSPI ADF,DB=16000
        MOV DRFY,NLIC
BCLTST: ADD QUATRE,ADSEUIL;SETMA              " LIRE COND.
        NOP
        DPX<DB;DB=ZERO          " DPX=0
        DB=MD                   " LIRE COD SUR DB
        BDBN LICOND             " ACCEPTER F LIAISON
        MOV ADF,ADF; SETMA;MI<DB;DB=DPX " INIT F
LICOND: DEC NLIC
        INC ADF;BGT BCLTST      " F SUIVANT, TST SUIVANT
"
SOMF2:  LDSPI SEPT;DB=7
        SUB SEPT,12
        DEC 12
        MOV 12,ADF2
        LDSPI INCFY;DB=0
        MOV DRFY,11             " SAV CPT
        LDSPI DRFY;DB=1
        LDSPI RFY;DB=15777
        MOV 11,NLIC
        JSR SOMFXY              " SOMME F SUR Y ATT2
"
NEGAT:  LDSPI 0;DB=16000        " F1=-F2
        LDSPI 1;DB=1
        LDSPI 2;DB=16000
```

```
        LDSPI 3;DB=1
        MOV 11,4
        JSR CVNEG
"
SOMF1:  MOV 12,ADF2
        DEC ADF2
        DEC ADF2
        LDSPI SEPT;DB=7
        LDSPI INCFY;DB=0
        LDSPI DRFY;DB=1
        LDSPI RFY;DB=15777
        MOV 11,NLIC
        JSR SOMFXY              " SOMME F SUR ATT1
"
        RETURN
"
        $END
```

ANNEXE 2

Nous décrivons ici des exemples qui montrent pour un type de module donné l'ensemble des premiers éléments qui constituent ses vecteurs.

1ERS ELEMENTS

VECTEURS MASSES	LIAISONS
MAS	KS
XS(n-2)	ZS
YS(n-2)	@XS1
XS(n-1)	@XS2
YS(n-1)	LGinit
XS(n)	aFX1
YS(n)	aFX2
FX	
FY	

CORDES	PEF
MAS	CALXent
LG	CALYent
XS1(n-2)	CALXsor
YS1(n-2)	CALYsor
XS2(n-1)	@VOIEXent
YS2(n-1)	@VOIEYent
XS1(n)	@VOIEXsor
YS1(n)	@VOIEYsor
.	FX
.	FY
KS1	XS(n-2)
ZS1	YS(n-2)
KS2	XS(n-1)
ZS2	YS(n-1)
KS3	XS(n)
ZS3	YS(n)

ANNEXE 3

Nous donnons ci-dessous les différentes équations qui décrivent les transformations d'unités : Utilisateur -> Univers de simulation. Soit le système d'unités utilisateur suivant décrit dans le système MKSA :
unité masse : rmksm;
unité de longueur : rmksl;
unité de temps : rmkst;
plage de variation physique des valeurs de conversion : 0,4096
plage de variation voulue par utilisateur :min, max

Les parmètres de simulation sont calibrés de la manière suivante:

```
masseS = masseU *rmksm;
raideurS = raideurU * rmkst**2;
frotS = frotU * rmkst;
xS = xU * rmksl;
yS = yU * rmksl;
x_transducteurS    =    (xtransducteur-2048)    *    (max-
min)*rmksl/4096;
f_transducteurS = (ftransducteur-2048) *  (max-min)*rmkst**2
/4096;
```

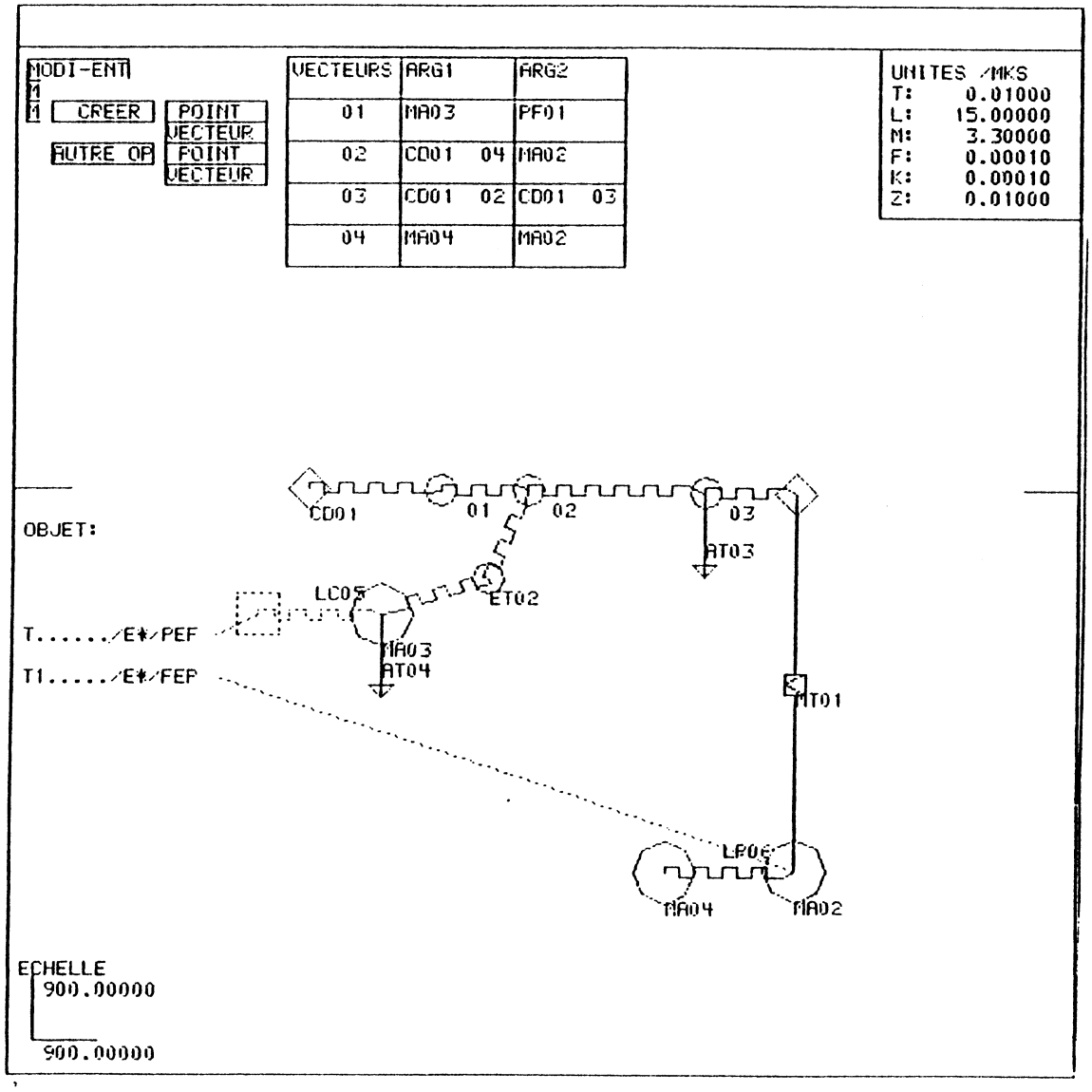
Annexe4

Liste des primitives et procédures de ANIGRAPH

gefilv0	dechaff	ecrcana	gomme
parcour	listmar	coef4	viewp
affsect	detseca	effasec	lirutl
lirutl	identse	inimat	vectet
cachmat	effmat	rempmat	identcase
effecra	poscase	posseim	edmatre
edmaten	edmatch	lirmaen	lirmare
lirmach	affmat	inimenu	contour
identfo	foncs	cachmen	effmenu
affmenu	rempmen	tlig	nbrli
fauxrou	afftxl	afftx	rempbuf
allsec	alltsec	alltfig	listobj
modsec	iniobj	fenetre	cloture
assimar	marqueu	msg	vism
encombr	rect	afvect	poscul
clipv0	posicur	liredi	convrl
rechmax	ajmaxpr	surf	feninter
traitid	detmaxp	ass2	assl
cachsec	inib	conver	code

Annexe 5

REPRESENTATION FIGURATIVE ET SYMBOLIQUE DANS ANIMA



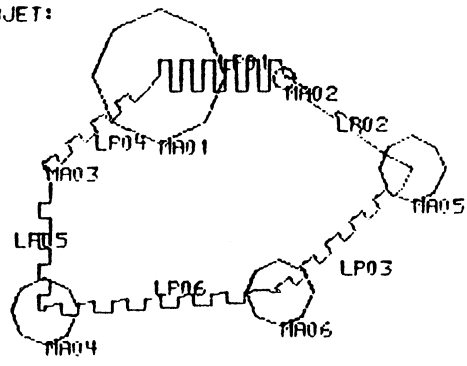
REPRESENTATION SANS FILTRAGE DANS ANIMA

P IMP/ MASSES
P IMP- SOLS
M LIST CORDES
LIST LI-PURS
LI-SOLS
ETOILES
LICS

UNITES /MKS
T: 0.01000
L: 0.00100
M: 0.10000
F: 0.00010
K: 0.00010
Z: 0.01000

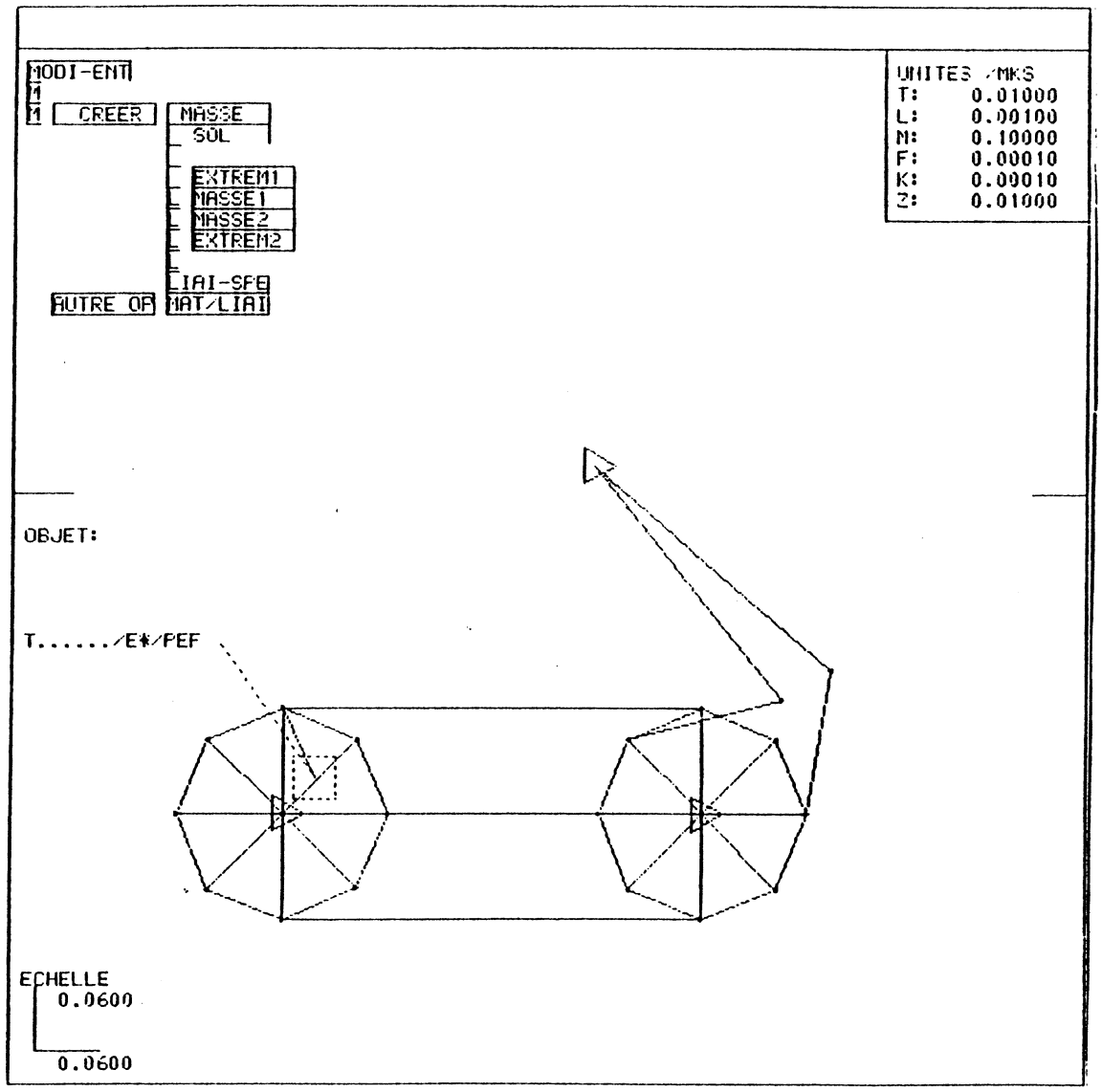
MASSES	X(*)	Y(*)	M
MA01	176	464	600
MA02	290	462	100
MA03	74	382	20
MA04	74	252	300

OBJET:

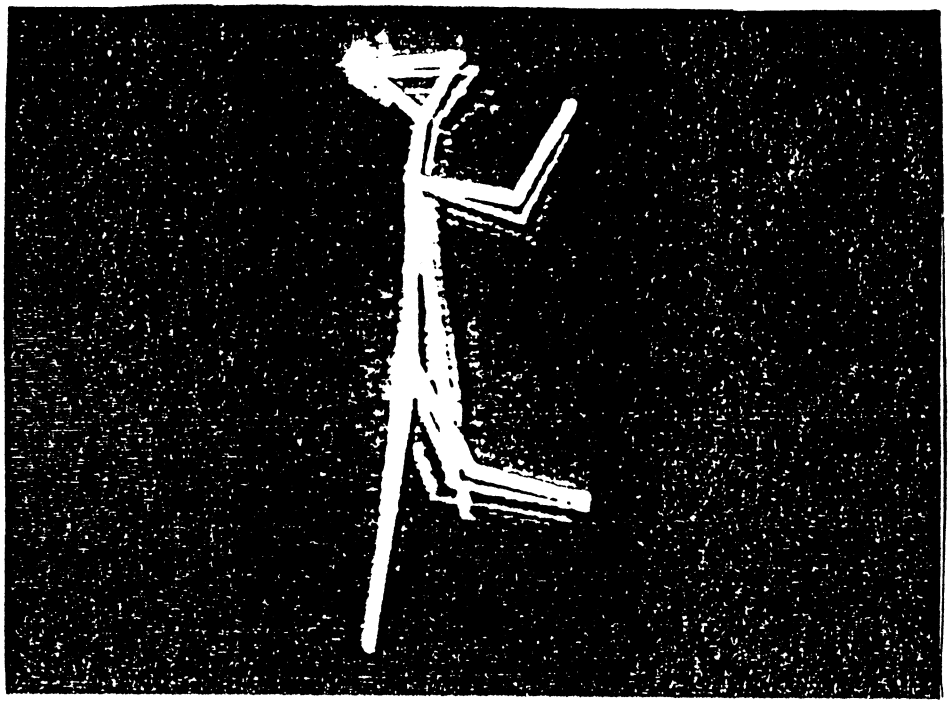


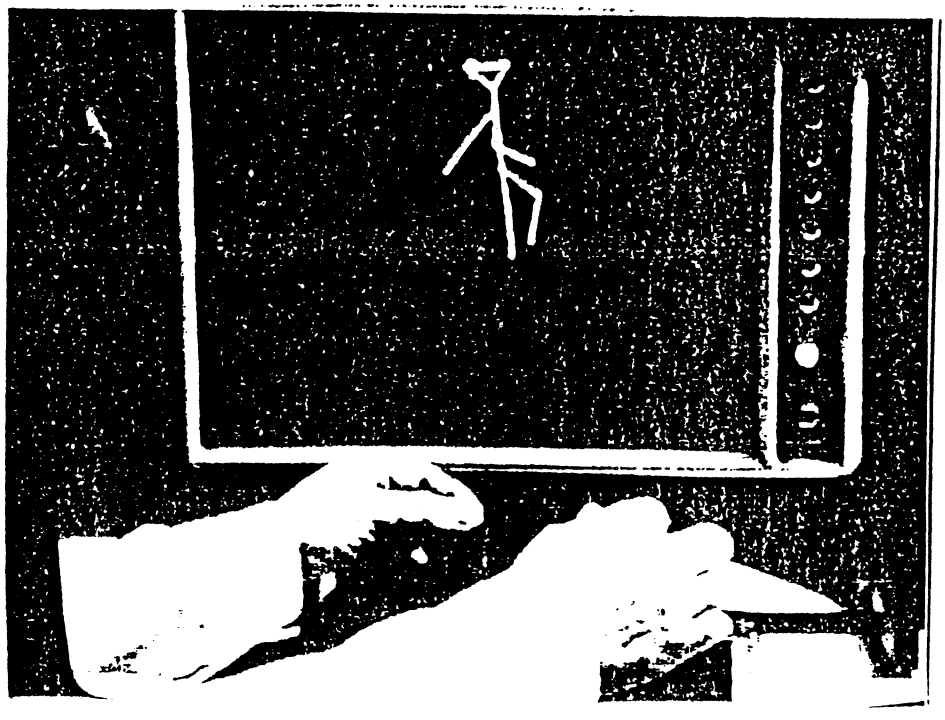
ECHELLE
0.0600
0.0600

REPRESENTATION AVEC FILTRAGE DANS ANIMA

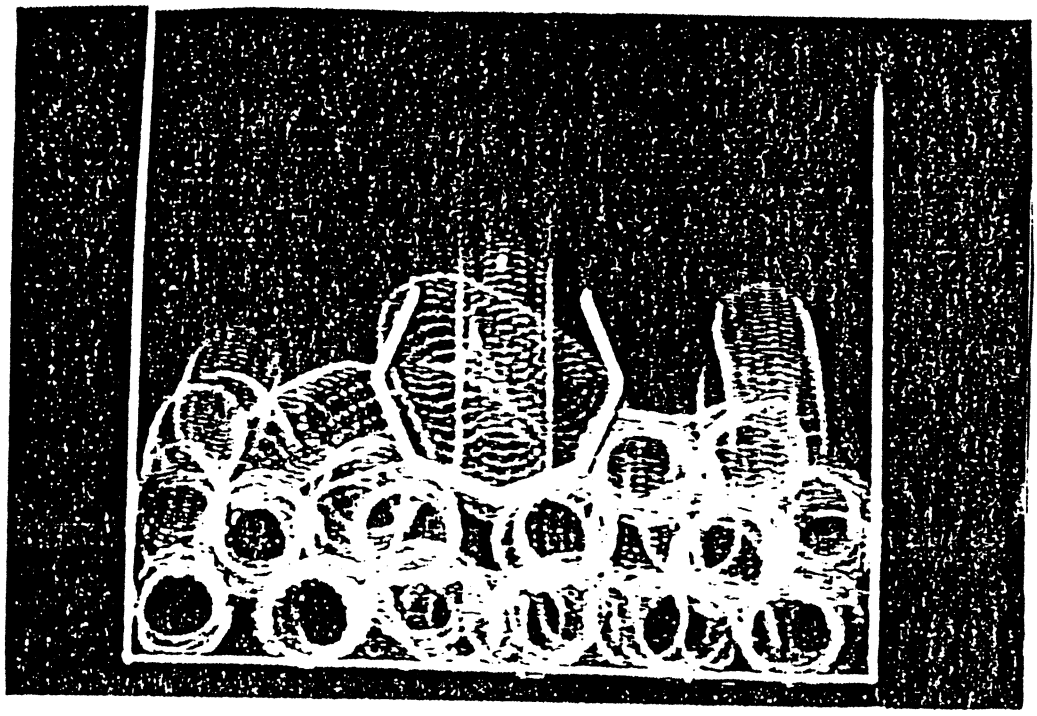


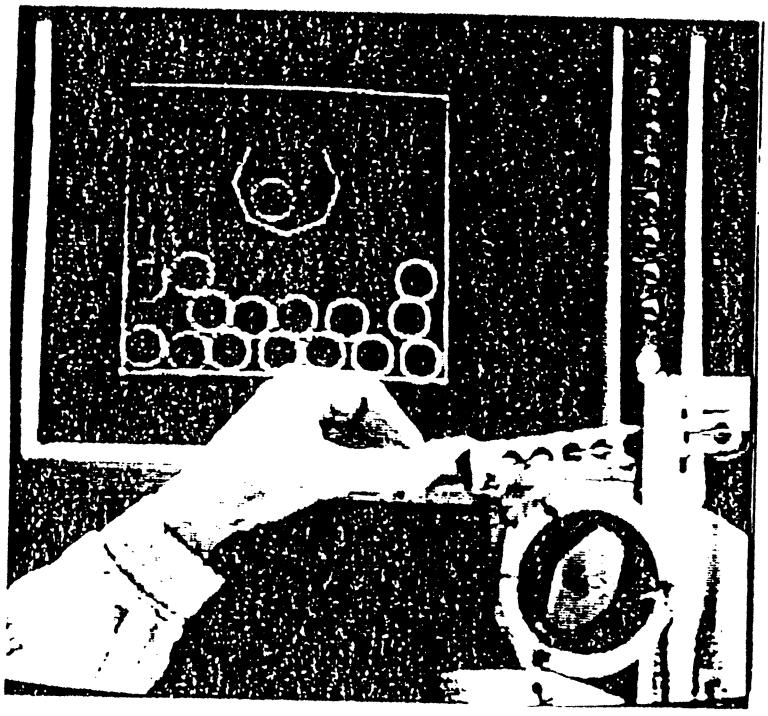
Annexe 6





Animation d'une marionnette par deux joysticks





Manipulation de billes par un transducteur gestuel rétroactif

BIBLIOGRAPHIE

1. ANIMATION, SYSTEMES D'AIDE A L'ANIMATION PAR ORDINATEUR

[1] B. ACKLAND, N. WESTE (1980)
Real time animation playback on a frame store dislay system.
SIGGRAPH proceedings.

[2] R. BAECKER (1979)
Digital video display systems and dynamic graphics.
SIGGRAPH proceedings.

[3] N. BURTNYK , M. WEIN (1971)
Computer generated key frame animation.
IEEE Congress.

[4] E. CATMULL (1980)
The problem of computer-assisted animation .
IEEE congress.

[5] G. COMPARETTI (1984)
Outil de production industrielle de dessins animés par ordinateur. Colloque image Biarritz.

[6] COUPIGNY (1979)
Fabrication assistée par ordinateur de dessins animés à l'aide du système Psyché-anim2.
Revue Radiodiffusion Télévision NO 57

[7] F.J. HONEY (1971)
Artist oriented computer animation.
SMPTE Journal

[8] K. JAHIDI (1984)

Application de la mécanique à l'animation interactive. Colloque image Biarritz.

[9] S. A. KALLIS (1971)
Computer animation techniques.
SMPTE Journal

[10] A. KITCHING (1977)
Antics : Graphic animation by computer.
SIGGRAPH Proceedings.

[11] M. J. LESTY (1977)
Étude d'un système informatique au service de l'artiste pour la réalisation de films à partir de croquis. Thèse 3eme cycle IMAG.

[12] A. LUCIANI (1985)
Un outil informatique de création d'images animées : Modèles d'objets, langage, contrôle gestuel en temps réel. Le système Anima.
Thèse Docteur-ingénieur INPG.

[13] F. MARTINEZ (1977)
Etude des problèmes de conception et de réalisation d'animation.
Le système Safran. Thèse 3eme cycle IMAG.

[14] J. C. MOISSENAC (1984)
Aide informatiques à la réalisation de dessins animés
Thèse 3eme cycle Saint-Etienne.

[15] M. SCHWEPPE (1984)
Motion in computer graphics. Colloque image Biarritz.

[16] Revue SCIENCES et TECHNIQUES (1984)
IMAGes de synthèse. Edition Mai 84.

[17] R. G. SHOUP (1979)
Color table animation.

SIGGRAPH proceedings.

[18] N. M. THALMANN, D. THALMANN (1985)
Computer Animation. Theory and practice.
Edition Tosiyasu L. Kunii.

2. COMMUNICATION HOMME-MACHINE, COMMUNICATION GRAPHIQUE

[19] E. ANSON(1979)
Semantics of graphical input. SIGGRAPH proceedings.

[20] E. ANSON (1980)
Device model of interaction
Workshop Seillac II.

[21] J. D. FOLEY (1980)
The art of natural graphic man-machine conversation.
IEEE Congress.

[22] Y. GARDAN (1982)
Elements méthodologiques pour la réalisation de systèmes de CFAO.
Thèse es sciences IMAG.

[23] P.F. JONES (1980)
Four principles of man-computer dialogue.
IEEE Congress.

[24] A. MORSE
Some principles for the effective d.brlay of data.
SIGGRAPH proceedings.

[25] ROSENTHAL et all
Detailed semantics of graphics input device.
SIGGRAPH Proceedings

3. SYSTEMES, SYSTEMES GRAPHIQUES, LOGICIELS GRAPHIQUES

[26] CROCUS (1975)
Systèmes d'exploitation des ordinateurs. DUNOD

[27] LEDUC-LEBAILLEUR (1977)
Etude et réalisation d'un logiciel graphique de base. Le logiciel Gri-Gri. Thèse 3eme cycle IMAG.

[28] ISO/DIS Draft international standard (1980)
Graphic Kernel System : Functional description

[29_a] M. LUCAS (1974)
Programmation des consoles de visualisation : les techniques de base.
USMG Laboratoire d'Informatique.

[29] LANG et all (1979)
Implementation of an interactive computer graphic environment at NASA/JSC.
SIGGRAPH proceedings.

[30] M. LUCAS (1979)
Réalisation des logiciels graphiques interactifs.
Edition Eyrolles

[31] M. LUCAS (mai 1984)
La synthèse d'image par ordinateur. Colloque image Biarritz.

[32] F. MARTINEZ (1982)
Approche systématique de la synthèse d'image. Thèse d'état IMAG.

[33] J. C. MARTY (1984)
Un éditeur graphique pour le système Cascade. Thèse 3eme cycle IMAG.

[34] W. NEWMAN, R. SPROULL (1979)
Principles of interactive computer graphics. Edition

[35] D. PILAUD (1982)
Méthode de conception descendante de système temps réel.
Thèse 3eme cycle. IMAG.

[36] V. G. SANCHEZ ARIAS (1985)
Un noyau pour la communication et la synchronisation de processus répartis.
Thèse docteur-ingénieur IMAG.

4. OUTILS DE PROGRAMMATION, METHODES DE PROGRAMMATION

[37] DIGITAL EQUIPMENT COORPORATION (1980)
LSI microcomputers and mémories.

[38] DIGITAL EQUIPMENT COORPORATION ()
RT-11 Advanced programmers guide.

[39] FLOATING POINT SYSTEM
- AP programmer reference manual Vol 1,2
- APMATH 38 vol 1,2 /4
- APLINK
- APLOAD
- IOP16

[40] OREGON MUSEUM OF SCIENCE AND INDUSTRY
- OMSI Pascal-1 V1.2/ RT11 User's Guid
- OMSI Pascal-1 V2.1/ RT11 User's Guide

[41] P. C. SCHOLL (1979)
Vers une programmation systématique: Etude de quelques méthodes, techniques et outils. Thèse es sciences IMAG.

5. MODELES THEORIQUES, SIMULATION MECANIQUE, RAPPORTS A.C.R.O.E

[42] T. DARS-BERBERYAN (1982)
Etude et réalisation d'un calculateur.brécialisé pour la synthèse sonore en temps réel par simulation de mécanismes instrumentaux.
Thèse Docteur-ingénieur INPG.

[43] C. CADOZ (1979)
Synthèse sonore par simulation de mécanismes vibratoires. Application aux sons musicaux. Thèse Docteur-Ingenieur INPG.

[44] P. LACORNERIE (1985)
Synthèse de sons par simulation des mécanismes instrumentaux : Logiciel pour le processeur Cordis-Temps-Réel. Thèse Docteur-ingénieur INPG.

[45] J. L. FLORENS (1978)
Coupleur gestuel interactif pour la commande et le contrôle de sons synthétisés en temps réel. Thèse Docteur-Ingenieur INPG.

[46] A. LUCIANI (1982)
Un outil de création d'images animées à l'aide de l'ordinateur : Concepts et premiere mise en oeuvre. Rapport ADI.

[47] A. LUCIANI (1982)
Un outil de création d'images animées à l'aide de l'ordinateur : Les grandes fonctions du système Anima. Rapport ADI

6. FILMOGRAPHIE DE DEMONSTRATION

[48] Bande audio-visuelle ACROE/LIFIA (ICMC 84)
[49] Bande audio-visuelle ACROE/LIFIA (Salon de la Musique 85) à paraître.

* Complement de bibliographie

[50] G. ALLAIN (1983)
Images de synthèse dans les simulateurs de vol.
Bulletin INRIA nO 88

[51] GUEDJ et ALL (1980)
Workshop Seillac II North-Holland

AUTORISATION DE SOUTENANCE

DOCTORAT ~~3ème CYCLE, DOCTORAT INGENIEUR, DOCTORAT USMG~~

Doctorat d'université (ancien régime)

Vu les dispositions de l'arrêté du 16 avril 1974,

Vu les dispositions de l'arrêté du 5 juillet 1984,

Vu les rapports de M. Annie Luciani

M. .....

M. Razafindrakoto Aimé est autorisé à présenter une thèse en vue de l'obtention du doctorat de l'université (ancien régime)

Grenoble, le 19 Décembre 1985

Le Président de l'Université Scientifique et Médicale

M. TANCHE

Mots-clés

Animation par ordinateur, expérimentation multisensorielle, processeur vectoriel, simulation mécanique, synthèse d'images, système interactif, temps réel, transducteurs.

Résumé

Partant de la spécification d'un univers d'objets à comportement mécanique, dynamique et visuel, le système ANIMA propose un langage alpha-graphique à l'utilisateur pour la description de ses objets, et le moyen d'expérimenter en temps réel leur animation. On étudie l'élaboration de ressouces indispensables à une phase compositionnelle, telles que mémoires de geste et mémoires d'images. Le système est vu comme la maquette d'un outil de création destiné à des artistes, construit autour d'un modèle général de simulation intégrant les comportements vibratoires et sonores des objets.