Utilisations de fenetres actives sous XWINDOW version 3.1

Thierry Viéville, Robert Fournier, Luc Robert

INRIA

UNITÉ DE RECHERCHE
INRIA-SOPHIA ANTIPOLIS

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports Techniques

N° 132

Programme 4
Robotique, Image et Vision

UTILISATIONS DE FENÊTRES ACTIVES SOUS XWINDOW VERSION 3.1

Thierry VIÉVILLE
Robert FOURNIER
Luc ROBERT et Cie

Septembre 1991

RT_0132

Utilisations de Fenêtres Actives sous Xwindow
Version 3.1

Thierry Viéville, Robert Fournier, Luc Robert, et Cie

7 août 1991

Résumé

Ce petit ensemble de routines permet de gérer des fenêtres interactives sous l'environnement *Xwindow*. On peut ainsi définir des fenêtres graphiques dans lesquelles vont s'afficher des images, des segments, des courbes et des portions de texte. La gestion de ces fenêtres, et la lecture des événements liés à la souris et au clavier est automatisée. De plus, divers comportements standards ont été programmés et sont directement utilisables. Il y a enfin des outils de séquencement d'événements, et de passage de messages.

Ce package est utilisé actuellement pour la réalisation de programmes interactifs en vision et robotique.

Using Active Windows in a Xwindow environment
Version 3.1

Abstract

This small package is used for the programming of interactive windows, in a Xwindow environment. One can define graphic windows in which pictures, segments, curves and texts are to be drawn. The management of these windows and the related input events from the pointer or the keyboard are automatic. In addition, some standard behaviors of such windows are preprogrammed and directly usable. At last, tools for events sampling and messages are available.

This package is now used for the design of interactive programs in vision and robotics.

Utilisations de Fenêtres Actives sous Xwindow
Version 3.1

Thierry Viéville, Robert Fournier, Luc Robert, et Cie

7 août 1991

Résumé

Ce petit ensemble de routines permet de gérer des fenêtres interactives sous l'environnement *Xwindow*. On peut ainsi définir des fenêtres graphiques dans lesquelles vont s'afficher des images, des segments, des courbes et des portions de texte. La gestion de ces fenêtres, et la lecture des événements liés à la souris et au clavier est automatisée. De plus, divers comportements standards ont été programmés et sont directement utilisables. Il y a enfin des outils de séquencement d'événements, et de passage de messages.

Ce package est utilisé actuellement pour la réalisation de programmes interactifs en vision et robotique.

Using Active Windows in a Xwindow environment
Version 3.1

Abstract

This small package is used for the programming of interactive windows, in a Xwindow environment. One can define graphic windows in which pictures, segments, curves and texts are to be drawn. The management of these windows and the related input events from the pointer or the keyboard are automatic. In addition, some standard behaviors of such windows are preprogrammed and directly usable. At last, tools for events sampling and messages are available.

This package is now used for the design of interactive programs in vision and robotics.

Utilisations de Fenêtres Actives sous Xwindow
Version 3.2

Thierry Viéville, Robert Fournier, Luc Robert, et Cie

4 septembre 1991

Table des matières

1 Introduction

Le petit package que l'on décrit ici permet de gérer des *fenêtres actives* sous l'environnement *Xwindow*. Il permet de définir des fenêtres graphiques dans lesquelles vont s'afficher des images, des segments, des courbes et des portions de texte. La gestion de ces fenêtres, et la lecture des événements liés à la souris est automatisée. De plus, divers comportements standards ont été programmés et sont directement utilisables.

L'avantage de cette approche est de rendre transparent à l'utilisateur toutes les opérations de gestion des fenêtres, et de lui fournir en standard des mécanismes sophistiqués au niveau de la souris, et du graphisme. Il y a en plus des outils de séquencement d'événements et de passage de messages.

Contrairement à l'usage, ce package n'utilise pas de *widgets*, mais a été construit directement sur la Xlib. Il va donc tourner sur tous les suns couleurs, et les terminaux X, et ne pose aucun problème de norme. Par contre, il est très spécifique et *très restrictif* :

(1) Les dialogues alphanumériques avec l'utilisateur se font simplement dans les fenêtres, les échanges de texte utilisent des scanf() et printf(), comme un programme standard, sur un terminal. Pas de menus déroulants, ni de fenêtre de dialogue, donc. Le graphique se fait dans la fenêtre X. Bien sûr, on peut toujours ajouter des widgets, il y a des bibliothèques déjà existantes qui le font.

(2) Le programme n'ouvre qu'une seule fenêtre, qui contiendra autant de sous-fenêtres que d'actions graphiques (affichages, boutons). La fenêtre principale ou **display** du programme est *de taille fixe*, les sous-fenêtres aussi, elle sont crées une fois pour toute. On pourrait modifier les tailles des fenêtres en cours d'exécution, mais ce n'est pas prévu actuellement, pour deux raisons : simplicté et rapidité.

(3) Les tracés se font en coordonnées réelles (1 pixel écran = 1 unité de tracé). Des fonctions permettent de passer à d'autres coordonnées. Une seule fonte est utilisée.

Donc, Ceci n'est pas un concurrent de *suntool* ou *gks*, ce n'est pas une *toolbox*, c'est quelques routines personnelles dont j'ai eu la patience d'écrire la doc. Si voulez faire du "graphique" utiliser les outils idoines.

La *philosophie* est de fournir un outil permettant de construire très vite et sans mise au point, des **programmes d'expérimentation d'algorithmes de vision**. En figeant plusieurs aspects de ces programmes, on économise du temps de reprogrammation. De plus, tous les programmes utilisant ce package auront un comportement similaire, il sera donc possible de les utiliser facilement. On tend aussi une petite main aux programmeurs sous suntools. En effet, j'ai essayé de reproduire toutes les fonctionnalités des programmes du projet robotvis qui tournent sous suntools, je l'ai rendu aussi "orienté objet" que possible. Par contre, cette bibliothèque n'est pas destinée à faire des programmes grand public. Sous X, il y a déjà plein de trucs pour ça.

Ce package permet donc de programmer rapidement et de manière transparente des objets graphiques actifs, sous X. Il reste complètement compatible avec le reste de l'environnement. C'est un outil très spécialisé, qui ne remplace pas les widgets déjà existants mais les complètent, avec des fonctions totalement nouvelles (Echantillonage, Rotations d'Images, etc...).

Remerciements : Ce package a été réalisé à partir des programmes de **Robert Fournier**, qui a largement contribué à cette réalisation. Un très gros merci à **Jean-Luc Szpyrka** pour l'installation de la Xlib sur Patmos, ce qui tient de la prouesse. Je voue une reconnaissance éternelle à **Daniel Terrer**, qui m'a constamment aidé et renseigné, homme des missions impossibles sur Patmos, et grand récupérateur des systèmes explosés.

Un **grand merci** à **Michel Buffa** et **Pascal Fua** qui ont donné des exemples de programmes, ont fournis les idées de base, ont aidé et conseillé en la matière. Merci aussi à **Régis Vaillant** et **Luc Robert** pour toutes les précieuses discussions concernant l'architecture et l'ergonomie de ce package.

Plusieurs routines ont été réalisées par des membres du projet robotvis ou grâce à leurs conseils. Certains interfaces ont été réalisés par des stagiaires. Tous sont cités au fur et à mesure.

Les routines de gestion de la machine patmos ont été réalisées grâce aux conseils de **Jean-Jacques Borrelly**, et l'intégration du système patmos est le fruit du travail de **Hervé Mathieu**.

Un très grand merci à **Théodore Papadopoulo** pour ces corrections, ses suggestions sur l'architecture, et son travail sur la documentation.

Distribution du package : Ce package est à la disposition de chacun, il suffit de le demander et de signer un contrat de mise à disposition. Il est gratuit, et a un mode de distribution similaire aux logiciels "gnu".

Ce package a déjà permis de réaliser une douzaine de logiciels de vision, il tourne depuis quatre mois. Il est utilisé au sein du projet robotvis. Le source de ce package sert de base à des programmes du projet Prisme.

Il permet de reprendre des programmes tournant sous suntools sous X.

Hors INRIA, il a été utilisé par **Uwe Schnepf** de G.M.B., Bonn et par **Christian Touseau** à l'E.S.M.E., Paris.

2 Fonctionnalités des fenêtres actives

Voici en détail, ce que savent faire les fenêtres actives, dans cette version.

2.1 Couleurs et niveaux de gris

La gestion de la **colormap** permet de visualiser des images à niveau de gris avec des tracés en couleur de courbes, segments et texte. Il permet de faire des superpositions en stéréoscopiques.

Les couleurs définies sont, en plus des 128 niveaux de gris :
Yellow Red Green Blue Cyan Magenta Brown
LightBlue LightGreen LightCyan LightRed LightMagenta
White Black Grey

La pseudo-couleur **InvVideo** conduit a l'affichage de l'objet par inversion video.

Les pseudo-couleurs **StrBlack, StrRed, StrGreen, StrWhite** permettent de faire des superpositions de segments selon les regles suivantes :
StrBlack sur **StrRed** = **StrRed**
StrBlack sur **StrGreen** = **StrGreen**
StrRed sur **StrGreen** = **StrWhite**
StrBlack et **StrWhite** sont inverses
StrRed et **StrGreen** sont inverses

Les *niveaux de gris* sont définis de manière complexe de façon à permettre d'y superposer des couleurs.

Lors de l'affichage sur des stations Monochromes, ou des stations couleurs statiques (Sun386) ou le système s'adapte au mieux.

Applications: visualiser des images avec des tracés en superposition.

Sur un sun monochrome : toutes les couleurs sont considérées comme du "blanc", seul la couleur noire est affichable. Le package reste utilisable.

2.2 Rotations de plans images

Une fenêtre peut avoir *plusieurs plans d'affichage*, plusieurs vues (16 au maximum). On peut donc charger plusieurs images et les visualiser successivement, en forme de film. La rotation d'image est automatisée.

Applications: voir un mouvement entre plusieurs images, ou le relief d'un objet 3D.

2.3 Gestion automatique de l'exposition

La mise à jour de la fenêtre qui réaffiche automatiquement tous ses éléments lors de mouvements, d'iconification, de superposition est faite de manière transparente.

Le système gère des *pixmaps* ou des *bitmaps* au niveau du serveur et pas au niveau du client (du programme lui même), ce qui accélère et simplifie considérablement la programmation, et l'exécution. Par contre ça bouffe de la place. Donc doucement sur le nombre de grandes fenêtres et le nombre de vues par fenêtre.

Applications: gestion automatique de tous les événements X-liés aux mouvements de fenêtre (Expose, Redraw, etc...).

2.4 Séquencement de tâches et envoi de messages

Chaque fenêtre est un objet actif, un *acteur*. A chaque fenêtre est associée une routine, un programme qui reçoit des événements et qui les traite. Il y a **pseudo-parallélisme** entre les tâches associées à chaque fenêtres. Cette tache est activée par trois types d'événements :

1. *La Souris*, et ses boutons. Si la souris se déplace ou si un bouton est pressé, la tâche de gestion de la fenêtre est appelée, avec une structure de donnée donnant tous les détails.
2. *Un Timer* qui peut activer périodiquement la tâche de gestion de la fenêtre. La période d'échantillonnage est en milli-secondes. Le temps est *à peu près réel* si l'on accepete l'usage du signal 14 Unix, en temps virtuel sinon. Dans tous les cas, le temps d'echantillonage est réellement de tant de milli-secondes quand le serveur X n'est pas surchargé. Si il y a des événements non-traités à temps, on les traite d'abord et on reprend le décompte du temps ensuite. C'est pas très joli, mais c'est un bon compromis : Cela correspond sur un sun à du temps CPU non utilisé par le serveur X, mais par le/les programme(s) eux mêmes. Chaque fenêtre a un et un seul timer, géré au niveau Unix, pas au niveau de la Xlib.
   Applications: Faire clignoter un objet graphique, simuler un mouvement, simuler un processus échantillonné dont on observe le comportement avec le graphique, évaluer - à la louche - le temps d'exécution d'un calcul.
3. *Des messages* envoyés d'une fenêtre *source* à une ou des fenêtres *cibles*. Une fenêtre peut envoyer à une autre fenêtre un message, qui va appeler la tâche de gestion de la fenêtre cible. Un message peut être envoyé à plusieurs fenêtres ou de plusieurs fenêtres. Le message est entièrement traité par la fenêtre cible avant de rendre la main à la fenêtre source, dans le cas d'une exécution sur un seul processeur. Très spécifiques de notre outil, ces messages n'ont rien à voir avec les messages de service du serveur X.
   Applications: Faire afficher dans une autre fenêtre, des paramètres d'une fenêtre active (position, courbe, objet graphique), simuler un processus recevant des interruptions, faire du pseudo-parallélisme.

2.5 Gestion de segments rémanents (Régis Vaillant)

Chaque fenêtre gère, pour chaque vue, une liste de segments rémanents, c'est à dire qui vont être mémorisés par la fenêtre et manipulés de manière automatique. Il y a similitude entre cette fonctionnalité et les "retained segments" de suncore.

Les segments après avoir été crées peuvent : s'afficher, clignoter, ne plus s'afficher, être selectionnés à la souris (comportent (**SelectSegment**)).

Applications: Sélectionner automatiquement à la souris des segments, les faire clignoter, les mémoriser.

2.6 Possibilité de créer des scénarios (Luc Robert)

Il est possible de mémoriser et de réexécuter une séquence d'actions de l'utilisateur (manipulation de la souris et du clavier). La gestion des événements mémorisés est entièrement automatique.

Applications: Cette fonctionnalité permet de conserver des démonstrations d'utilisation des programmes créés par l'utilisateur du package.

2.7 Simulation d'un oscilloscope numérique.

Il est possible de gérer une ou des trace d'oscilloscope dans une fenêtre. Au fur et à mesure de leur envoi à la fenêtre les échantillons sont affichés et la trace décalée automatiquement.

La surface graphique sur laquelle est affichée la trace est sauvegardée et restaurée au fur et à mesure. Une fenêtre active peut gérer jusqu'à 8 traces de scope.

2.8 Transformation de coordonnées

Les tracés se font en coordonnées réelles (1 pixel écran = 1 unité de tracé). Ceci permet de gérer à une vitesse maximale les tracés sur l'écran.

Il existe des fonctions qui permettent de passer à d'autres types de coordonnées. On appelera coordonnées 1D le simple changement d'échelle sur l'axe horizontal et vertical, coordonnées 2D une transformation affine du plan d'affichage (avec translation, rotation et changement d'échelle), coordonnées 3D une projection perspective (avec translation, rotation et changement d'échelle) d'un point 3D.

2.9 Comportements prédéfinis

Chaque fenêtre peut avoir des **comportements particuliers** préprogrammés. On en mettra de plus en plus; les premiers déjà définis sont :

- (**UsingView(n)**) déclare combien de vues vont être gérées par la fenêtre active. L'entier n va de 0 à 16. On recommande $n == 1$, par défaut.
  - si $n == 1$ la fenêtre ne gère qu'une vue et effectue automatiquement les réexpositions.
  - si $n == 0$ la fenêtre ne gère pas de vue, et ne fait pas de réexposition.
- (**ReadRectangle**) avec le bouton gauche de la souris. Quand l'utilisateur presse le bouton et déplace la souris il crée un rectangle de taille variable qui se fige quand le bouton est relâché. Les coordonnées du rectangle ainsi définies sont disponibles.
  Applications : lecture d'une fenêtre d'intérêt dans une image, tracé d'un objet graphique inclus dans un rectangle, sélection d'un objet en indiquant sa position.
- (**ReadLine**) avec le bouton gauche de la souris. Quand l'utilisateur presse le bouton et déplace la souris il crée une ligne de taille variable qui se fige quand le bouton est relâché. Les coordonnées de la ligne ainsi définies sont disponibles.
  Applications : lecture d'une position et orientation dans une image, trac'e d'un objet graphique sélection d'un objet en indiquant sa position, et sa longueur.
- (**ReadSemiLine**) avec le bouton gauche de la souris. Quand l'utilisateur presse le bouton et déplace la souris il crée une petite fleche de taille fixe qui se fige quand le bouton est relâché. On définit ainsi un point et une direction de demi-droite. La direction en radians est donnée par la macro SemiLineAngle().
  Applications : lecture d'une position et orientation dans une image, sélection d'un objet en indiquant sa position, et son orientation.
- (**ReadReticule**) avec le bouton gauche de la souris. Quand l'utilisateur presse le bouton et déplace la souris il crée un réticule qui indique le point courant et qui se fige quand le bouton est relaché. Les coordonnées du point ainsi chois sont disponibles.
  Applications : lecture d'une position dans le fenêtre, sélection d'un objet, etc...
- (**ActAsButton**). Quand un des trois boutons est enfoncé, la fenêtre flashe, se met en vidéo inverse. Elle revient en position normale lorsque le bouton est relaché.
  Applications : indication qu'un bouton est enfoncé, création de fenêtres de type "menu".
- (**RotateViews**) avec le bouton droit de la souris. Quand l'utilisateur presse le bouton, les différents plans de l'image, les vues, sont affichés successivement à chaque pression du bouton.
- (**SelectSegment**) avec le bouton gauche de la souris. Quand l'utilisateur presse le bouton, et le relâche, le segment rémanent le plus proche du curseur est sélectionné.

2.10 Gestion simplifiée des événements XWindow

Les événements du serveur X sont à peu près tous gérés par défaut, pour la plus grande joie de l'utilisateur. Les comportements décrits précédemment captent les événements qui les concernent, agissent en conséquence et passent la main à la tâche de gestion de la fenêtre ensuite.

A chaque appel de la tâche de gestion de la fenêtre un pointeur vers une structure de donnée est passé. Cette structure de donnée contient toutes les infos utiles pour traiter l'événement.

Voici les événements qui sont envoyés à la tâche de gestion d'une fenêtre :

- **(Starting)** : Cet événement est émis une fois, au lancement de la fenêtre, il permet à la tâche de gestion de la fenêtre de faire les tracés de départ, d'initialiser le graphique et les variables de la fenêtre. Chaque fenêtre ayant son propre programme de gestion, le code correspondant à cet événement est en quelque sorte le main() pour la fenêtre. Cela correspond à l'événement MapNotify de la Xlib (mais il est géré différemment).
- **(Button1Pressed, Button2Pressed, Button3Pressed, Button1Released, Button2Released, Button3Released)** : Un des trois boutons vient d'être enfoncé ou relaché. La tâche peut alors réagir, et faire quelque chose. Ce sont les événements ButtonPressed et ButtonReleased de la Xlib, le bouton concerné ayant été reconnu. La macro **ButtonReleased** utilisée avec la syntaxe :
  ```
  case ButtonReleased:
  ```
  est équivalente à :
  ```
  case Button1Released:
  case Button2Released:
  case Button3Released:
  ```
- **(Motion)** : La souris vient d'être déplacée vers une nouvelle position ou elle vient de sortir ou entrer de la fenêtre. Sa position dans la fenêtre ou au bord est donnée par (Win→x, Win→y).
- **(Sampling)** : Le timer vient de détecter que la période d'échantillonnage est écoulée. Il appelle la tâche de gestion à ce titre.
- **(Message(nn))** : Un message vient d'être reçu pour cette fenêtre. Le numéro du message est *nn*, il peut varier de 0 a 256, un pointeur vers le message est donné par Win→buffer. Le message est donc contenu dans un tableau de la forme :
  ```
  char Win->buffer[];
  ```

Exemple : Voici un exemple d'application : pour faire clignoter un objet (1 coup sur 2) utiliser comme tâche de gestion de la fenêtre :

```
TacheDeGestionDeLaFenetre(Win)
ActiveXWindow *Win;
{
   /* On va traiter les evenements en fonction de leur source */
   switch(Win->Event) {
    case Sampling:       /* On s'interesse aux evenements du timer */
        if(odd(Win->localtime)) { /* On traite alternativement A ou B */
         ... A ...
        } else {
         ... B ...
        }
     break;
    case .../...        /* On traite d'autres evenements */
   }
}
```

qui exécutera alternativement les deux blocs A et B.

3 Programmer des fenêtres actives

La programmation de fenêtres actives se fait en trois blocs :

(a) Un programme principal qui lit les arguments d'appel, déclare et ouvre les fenêtres, passe la main au noyau de ce package.

(b) Une tâche de gestion par fenêtres ouvertes dans le display de l'application. Ces tâches n'ont pas de boucles. Elle reçoivent un événement, le traite et rendent la main.

(c) Toutes les autres fonctions et procédures de calcul et de traitement du programme.

3.1 Le programme principal

Le programme principal principal assure quatre fonctions :

1. Il lit les paramètres passés comme arguments au programme. On recommande de passer un maximum de paramètres (noms des fichiers, options, paramètres numériques) comme argument au programme. Cette méthode permet à l'utilisateur de dialoguer au niveau du shell d'Unix (plutôt qu'à travers de sordides printf() et scanf() ou de compliqués menus), d'inclure votre programme dans des shell-scripts. C'est standard, ça évite de longs dialogues à la souris et au clavier. On présente en annexe quelques routines de gestion des paramètres du programme. Très pratiques, elles facilitent la lecture des paramètres du programme.
2. Il déclare et ouvre la fenêtre principale, le display du programme. Pour cela on appelle la procédure :

3.1.1 InitXDisplay()

- Cette routine doit être appelée **une fois au début du programme.**
- La taille du display doit être suffisante pour contenir toutes les fenêtres de travail déclarées ensuite.

```
/*-----------------------------------------------------------------*/
InitXDisplay(x0, y0, w, h, label)
int x0, y0, w, h;              /* Position (x0,h0) sur l'ecran
                                  et taille (w,h) du display en pixel */
char *label;                   /* Nom de la fenetre */

 Si x0=y0=0 la fenetre est postionnee par le window manager.

/*-----------------------------------------------------------------*/
```

3. Il déclare et ouvre toutes les fenêtres de l'application. Pour chaque fenêtre on appelle MakeXWindow() qui renvoie un pointeur vers la structure de donnée attachée à la fenêtre.

3.1.2 MakeXWindow()

```
/*-----------------------------------------------------------------*/
ActiveXWindow *MakeXWindow
 (x0, y0, w, h, Behavior, SamplingPeriod, EventTask)
int x0, y0, w, h;              /* Position (x0,h0) dans le display du
                                  programme et taille (w,h) de la fenetre */
int Behavior;                  /* Flag decrivant le comportement de la
                                  fenetre */
int SamplingPeriod;            /* Periode d'echantillonage en
                                  milli-secondes, elle sera approximee
                                  par un multiple de la periode
                                  d'horloge du programme (50msec) */
void (*EventTask) ();          /* Tache de gestion des evenements */
```

```
 A defaut d'un besoin specifique il est conseille de choisir :

Behavior = UsingViews(1)
SamplingPeriod = 0
EventTask = 0

/*-----------------------------------------------------------------*/
```

• L'entier Behavior est un "ou logique" entre un sous-ensemble de : UsingViews(i), ReadRectangle, ReadLine, ReadSemiLine, ReadReticule, ActAsButton, RotateViews, SelectSegment, selon les besoins.
Les comportements ReadRectangle, ReadLine, ReadSemiLine, ReadReticule ActAsButton, et SelectSegment sont incompatibles, si les trois sont définis simultanément un seul sera réellement pris en compte.
• La procédure EventTask est définie par l'utilisateur.

4. Il démarre le noyau de notre package. Il suffit d'appeler la routine *StartActiveXWindows();*.

3.1.3 GereActiveXWindows() Exit() AwExit()

GereActiveXWindows() est la routine qui enclenche le contrôle des événements intervenants au niveau des fenêtres actives. Contrairement à StartActiveXWindows() cette routine ne rend pas la main mais boucle indéfiniment.
AwExit() termine la gestion des fenetres de l'echantillonage et des evenements graphiques, et provique l'arret de GereActiveXWindows(). L'exécution du programme reprend alors apres GereActiveXWindows().

```
/*-----------------------------------------------------------------*/
GereActiveXWindows()

 declenchent  l'echantillonage et la gestion de fenetres actives.

AwExit()

 arrete l'echantillonage et la gestion de fenetres actives, et provoque
l'arret de GereActiveXWindows().
/*-----------------------------------------------------------------*/
```

Exemple : Voici un exemple de programme principal

```
#include <stdio.h>
#include <math.h>
#include "activewin.h"

ActiveXWindow *Win1, *Win2, *Win3;      /* On utilisera 3 fenetres */
extern void (* Win1EventTask)(),
            (* Win2EventTask)(),
            (* Win3EventTask)();        /* On a defini ailleurs les
                                           taches de gestion des 3
                                           fenetres */
main()
{

/* 1) On ouvre le display */
```

```
  InitXDisplay(0, 0, 1100, 512, "xvis2");        /* On ouvre un display
 -> en haut a gauche de l'ecran (0,0),
 -> de 1100 pixels de large et 512 de haut,
 -> dont le nom reconnu par le serveur X est "xvis2"
*/

/* 2) On ouvre les fenetres */

  Win1 = MakeXWindow(0, 0, 512, 512,
   UsingViews(3) | RotateViews | ReadReticule,
   50, Win1EventTask);                          /* On ouvre une fenetre
 -> en haut a gauche du display,
 -> de 512 x 512 pixels,
 -> qui aura 3 plans de vue, saura les fera tourner, et lire un reticule
 -> qui sera echantillonnee toutes les 50 msec,
 -> dont la routine de gestion est Win1EventTask()
*/

    Win2 = MakeXWindow(512, 0, 512, 512,
     UsingViews(1) | ReadRectangle, 3000, Win2EventTask);        /* idem */

    Win3 = MakeXWindow(1024, 10, 70, 30,
     UsingViews(1) | ActAsButton, 0, Win3EventTask);             /* idem */

/* 3) On lance la gestion d'evenements */

    GereActiveXWindows();
}
```

3.2 La tâche de gestion pour chaque fenêtre

La tâche de gestion a une structure très simple comme indiqué ci dessous. Il en faut une par fenêtre, passé en argument lors du MakeXWindow() :

```
 /* Modele de routine de gestion d'evenements d'une fenetre active */
void EventTask(Win)
ActiveXWindow *Win;
{
  switch (Win->Event) {
  case Starting:
    --- on traite ici le fait que la fenetre apparait ---
    break;
  case Sampling:
    --- on traite ici le fait qu'il y ait une periode ecoulee ---
   break;
  case Motion:
   --- on traite ici le fait qu'il y ait mouvement de souris ---
     break;
  case Button1Pressed:
   --- idem ---
    break;
  case Button2Pressed:
   --- idem ---
```

```
    break;
  case Button3Pressed:
   --- idem ---
    break;
  case Button1Released:
   --- idem ---
    break;
  case Button2Released:
   --- idem ---
    break;
  case Button3Released:
   --- idem ---
    break;
  }
}
```

3.2.1 Passage d'options en parametres : InitXDisplayOptions()

On peut faire prendre en compte des options de manière automatique à l'aide de la routine suivante, qui doit être appellée **avant** InitXDisplay().

```
/*-----------------------------------------------------------------*/
InitXDisplayOptions(argc, argv)
int argc;
char *argv[];

analyse la ligne de parametres passes au programme et prend en compte
les options correspondantes. Les options actuellement prises en compte
sont :

 -display display-name      qui permet de communiquer et d'afficher a
                            travers le reseau.

 -geometry +x0+y0           qui permet de positionner la fenetre, la
                            taille de la fenetre n'est -par principe-
                            pas ajustable, cette option surpasse les
                            parametres x0,y0 passes a InitXDisplay().

 -bgcolor [Red|Green|Blue|Brown|Yellow|Grey|Cyan|Magenta|
          LightRed|LightGreen|LightBlue|LightCyan|LightMagenta|
          Black|White]      qui permet de marquer d'une couleur le fond de
                            la fenetre.

 -static                    qui force la colormap a etre statique.

 Les parametres (argc, argv) sont d'abord stockes dans les variables
globales :

int xargc;
char **xargv;

et ensuite traitees avec les routines de l'annexe B.1. Les options
restantes non traitees par InitXDisplayOptions() sont accessibles
dans ces variables globales. Une utilisation typique est de la
```

forme :

```
extern int xargc;
extern char **xargv;

main(argc, argv)
int argc;
char *argv[];
{

  /* Lecture des parametres du programme */
  {
    int ErreurParametre = 0;

    InitXDisplayOptions(argc, argv);

    ErreurParametre = (xargc != 1);

    if (ErreurParametre) {
      printf("\nUsage: ...
      exit(-1);
    }
  }

  /* Lancement du display */
  InitXDisplay(0, 0, ...

 .../...
}

/*-----------------------------------------------------------------*/
```

3.3 Fenêtres et Comportements prédéfinies

3.4 La gestion des comportements prédéfinis

3.4.1 Relecture de paramètres

On notera que l'événement (**Button1Released**) correspond, dans les cas des comportements (**ReadRectangle**), (**ReadLine**), (**ReadSemiLine**), (**ReadReticule**) ou (**SelectSegment**), au moment où l'utilisateur du programme a fini de spécifier le rectangle ou la position. Et l'information est alors disponible, la signification des paramètres est décrite Fig.1.

(**ReadRectangle**) : Le rectangle a ces deux coins donnés par (Win→x, Win→y) et (Win→x1, Win→y1).

(**ReadReticule**) Le réticule a sa position donnée par (Win→x, Win→y).

(**ReadLine**) La portion de ligne sélectionnée a ces deux extrémités données par (Win→x, Win→y) et (Win→x1, Win→y1). On obtient son angle avec la macro SemiLineAngle(Win) et sa longueur par la longueur LineLength(Win).

(**ReadSemiLine**) La demi-droite sélectionnée a son extrémitée donnée(Win→x1, Win→y1) et son angle donné par SemiLineAngle(Win).

(**SelectSegment**) Le segment sélectionné a ces deux extrémités données par (Win→x, Win→y) et (Win→x1, Win→y1) et son index égal a Win→i. Le segment sélectionné est le plus proche de la position de la souris lors de la sélection, et Win→i vaut 0 si on aucun segment

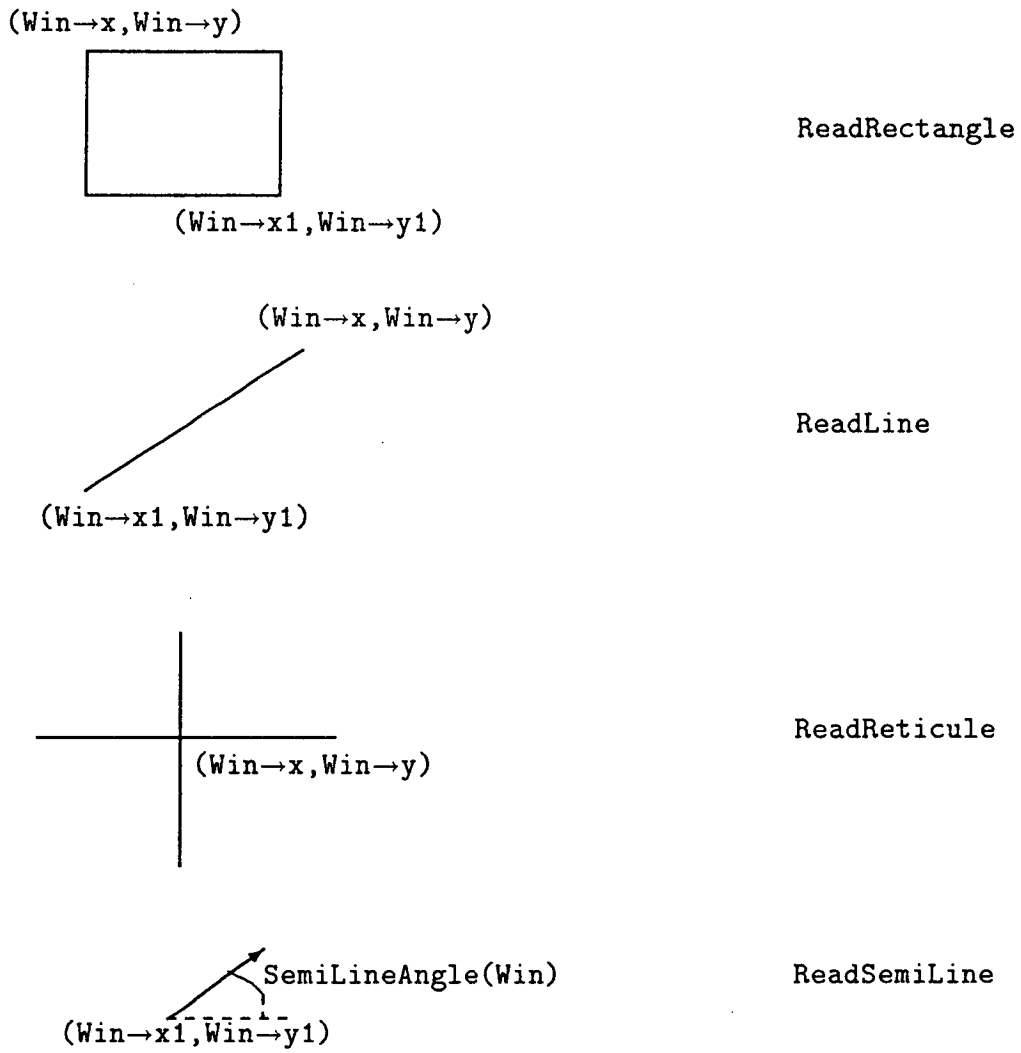


Figure 1: Description des paramètres renvoyés au Button1Released

n'est dans le voisinage de la souris lors de la sélection.

3.4.2 Comportements automatiques

(**UsingView(n)**) se fait automatiquement et ne nécessite aucun traitement de la part de l'utilisateur.

(**ActAsButton**) se fait automatiquement et ne nécessite aucun traitement de la part de l'utilisateur.

(**RotateViews**) se fait avec le bouton droit de la souris. Quand l'utilisateur presse le bouton, les différents plans de l'image, les vues, sont affichés successivement à chaque pression du bouton.

3.5 Quelques fenêtres prédéfinies

3.5.1 Bouton de commande : MakeXButton() DrawXButton()

Cette fonction ouvre une fenêtre sur fond blanc qui affiche la chaine de caractères Label en son milieu, puis appelle EventTask() quand un bouton est appuyé ou relaché. On obtient ainsi un bouton qui exécute une action lorsqu'on le sélectionne.

```
/*-------------------------------------------------------------------*/
ActiveXWindow *MakeXButton
 (x0, y0, w, h, Label, EventTask)
int x0, y0, w, h;               /* Position (x0,h0) dans le display du
                                   programme et taille (w,h) du bouton */
char Label[];                   /* Ente^te du bouton */
void (*EventTask) ();           /* Tache de gestion des evenements */
/*-------------------------------------------------------------------*/
/*-------------------------------------------------------------------*/
DrawXButton(Win, color, label)
ActiveXWindow *Win;
int color;
char label[];

 permet de reecrire le bouton sur un fond colore. Utile pour ge'rer
des boutons dynamiques, dont la fonction change au cours du temps.

Exemple :

MoveStop(Win)
ActiveXWindow *Win;
{
  static int move_else_stop;

  switch (Win->Event) {
  case Starting:
    move_else_stop = 1;
    DrawXButton(Win, Yellow, "Move");
    break;
  case Button1Released:
    if (move_else_stop) {
      move_else_stop = 0;
      DrawXButton(Win, Yellow, "Stop");
    } else {
      move_else_stop = 1;
      DrawXButton(Win, Yellow, "Move");
    }
    break;
  }
}

.../...

 MakeXButton(x0, y0, w, h," ", MoveStop);

/*-------------------------------------------------------------------*/
```

3.5.2 Sélection d'une couleur : MakeXPalette() (Luc Robert), colorname() colorlabel()

Cette fonction ouvre une fenêtre qui affiche la palette des couleurs et permet la sélection d'une couleur en fonction avec le curseur de la souris. Pour sélectionner une couleur il suffit d'aller cliquer sur la couleur désirée; la couleur une fois sélectionnée la tache WinTask est appellée. La gestion de la fenêtre de sélection des couleurs est automatique.

La couleur lue est stockée dans la variable Win− >bgcolor.

```
/*-----------------------------------------------------------------*/
ActiveXWindow *MakeXPalette(x0, y0, w, h, WinTask)
int x0, y0, w, h;              /* Position (x0,h0) dans le display du
                                  programme et taille (w,h) de la palette */
void (*WinTask) ();          /* Tache de gestion de la fenetre qui
                                recupere la valeur entree par
                                l'utilisateur */

 A priori une taille de 64 x 64 (w=64, h=64) est recommandee.

/*-----------------------------------------------------------------*/
```

Il est aussi possible de choisir une couleur ou d'imprimer son nom à partir d'un programme :

```
/*-----------------------------------------------------------------*/
int colorlabel(i)
int i;

 Renvoie pour un entier modulo 15 le code d'une des 15 couleurs de
la palette. Utilise pour colorie differement un ensemble d'objets; il
suffir de l'afficher avec la couleur colorlabel(index) ou l'entier
index numerote l'objet.
/*-----------------------------------------------------------------*/
/*-----------------------------------------------------------------*/
char *colorname(color)
int color;

 Renvoie pour une couleur donne un pointeur vers une chaine de
caracteres correspondant au nom de la couleur. Attention la chaine
de caracteres est modifiee a chaque appel de cette fonction.
/*-----------------------------------------------------------------*/
```

3.5.3 Sélection d'un angle ou d'une position : MakeXVolantButton(), MakeXCurseurButton(), MakeXUCurseurButton(), MakeXEditButton() MakeMenuButton() (Luc Robert)

Ces fonctions ouvrent une fenêtre qui affiche un volant (MakeXVolantButton()) ou une réglette (MakeXCurseurButton() et MakeXUCurseurButton()) ou une fenêtre d'édition sur une ligne, et permettent la sélection, par l'utilisateur, d'un angle ou d'une position linéaire, ou l'entrée d'un paramètre. MakeMenuButton() permet de definir un menu en ASCII.

- Le mécanisme de MakeXCurseurButton() et MakeXUCurseurButton() est le suivant :
  - Le bouton **gauche** déplace le curseur vers la gauche.
  - Le bouton **droit** déplace le curseur vers la droite.
  - Le bouton **central** déplace le curseur à la position où il se trouve.

  La valeur du curseur varie entre -1.0 (curseur à gauche), 0.0 (curseur au centre) et 1.0 (curseur à droite). La valeur courante est dans la variable Win→a.
- Le mécanisme de MakeXVolantButton() est le suivant :

- Le bouton **gauche** déplace le volant dans le sens trigonométrique.
- Le bouton **droit** déplace le volant dans le sens des aiguilles d'une montre.
- Le bouton **central** déplace le volant à la position où il se trouve.

La valeur de l'angle varie entre -Pi et Pi, et vaut 0.0 pour un volant horizontal orienté vers la droite de l'écran. La valeur courante est dans la variable Win→a.

- Le mécanisme de MakeXEditButton() est le suivant :
  - Les caratères sont entrés 1 par 1 par l'utilisateur et lors du RETURN la chaine de caractère est acceptée est un évènement de type **ButtonReleased** est généré.

  La valeur courante est pointé par la variable Win→buffer.
- Le mécanisme de MakeXMenuButton() est le suivant :
  - Chaque item du menu correspond a un caratère du clavier. Lorsque l'utilisateur frappe une touche un évènement de type **KeyPressed** est généré, et le caractère choisit est dans le paramètre Win→i.

La gestion de la fenêtre se fait par une routine identique à la gestion d'une fenêtre ordinaire et qui traite l'événement **ButtonReleased**, qui correspond aux moments où l'utilisateur a changé le volant ou le curseur.

Ces fonctions ont été imaginées et réalisées par Luc Robert.

```
/*-----------------------------------------------------------------*/
ActiveXWindow *MakeXVolantButton(x0, y0, w, h, title, color, WinTask)
ActiveXWindow *MakeXCurseurButton(x0, y0, w, h, title, color, WinTask)
ActiveXWindow *MakeXUCurseurButton(x0, y0, w, h, title, color, WinTask)
ActiveXWindow *MakeXEditButton(x0, y0, w, h, title, color, WinTask)
ActiveXWindow *MakeXMenuButton(x0, y0, w, h, title, color, WinTask)
int x0, y0, w, h;            /* Position (x0,h0) dans le display du
                                programme et taille (w,h) du bouton */
char title[];                /* Titre de la fenetre qui est affichee
                                a cote du bouton */
int color;                   /* Couleur du bouton */
void (*WinTask) ();          /* Tache de gestion de la fenetre qui
                                recupere la valeur entree par
                                l'utilisateur */

Exemples d'utilisation :

 [1] Dans le programme suivant on a definit un curseur positionne
en (67,0) de taille (64,64) de titre "toto", de couleur LightBlue,
gere par la routine CurseurTask().

 La routine CurseurTask() recupere l'evenement ButtonReleased et se
contente d'afficher la valeur Win->a donnee par l'utilisateur.

*************************************************************************
#include <stdio.h>
#include <math.h>
#include "activewin.h"

ActiveXWindow *WinT;


CurseurTask(Win)
ActiveXWindow *Win;
{
```

```
  switch (Win->Event) {
  case ButtonReleased:
    Awprintf(WinT,"Curseur=     break;
  }
}

.../...

MakeXCurseurButton(67, 0, 64, 64, "toto", LightBlue, CurseurTask);

.../...
```

****

[2] Dans le programme suivant on a definit une fenetre d'edition positionnee en (67,0) de taille (100,30) de titre "toto", de couleur LightBlue, geree par la routine EditTask().

La routine EditTask() recupere l'evenement ButtonReleased et se contente d'afficher la valeur donnee par l'utilisateur.

****

```
#include <stdio.h>
#include <math.h>
#include "activewin.h"

ActiveXWindow *WinT;

EditTask(Win)
ActiveXWindow *Win;
{
  switch (Win->Event) {
  case ButtonReleased:
    Awprintf(WinT, "Edit=    break;
  }
}

.../...

MakeXEditButton(67, 0, 100, 30, "toto", LightBlue, EditTask);

.../...
```

****

[3] Dans le programme suivant on a definit une fenetre de menu positionnee en (67,0) de taille (100,30), de couleur LightBlue, geree par la routine MenuTask().

La routine MenuTask() recupere l'evenement KeyPressed et se contente de ge'rer les items 'f' et 'q'. Le menu est definit par le titre de la fenetre.

```
*************************************************************************
#include <stdio.h>
#include <math.h>
#include "activewin.h"

ActiveXWindow *WinT;

EditTask(Win)
ActiveXWindow *Win;
{
  switch (Win->Event) {
  case KeyPressed:
   switch ((char) Win->i) {
    case 'q':
     Exit(0);
    case 'f':
     InvertXColors(Win);
     break;
    } break;
  }
}

.../...

MakeXMenuButton(67, 0, 100, 30, "(f)lash\n(q)uit", LightBlue, EditTask);

.../...

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

/*-----------------------------------------------------------------*/
```

3.5.4 Gestion de scenarios : MakeXEditScenario() (Luc Robert)

Il est possible de creér une fenêtres qui éditer et stocke et exécute des scénarios. Il y a trois actions définies :

```
/*-----------------------------------------------------------------*/

> Les trois actions :

 Record : qui permet de memoriser un scenario.
  Presser le bouton du milieu pour commencer la memorisation.
  Presser de nouveau le bouton du milieu pour finir la memorisation.

 PlayAll : qui permet d'executer en continu un scenario.
  Presser 2 fois le bouton du milieu pour commencer l'execution.

 PlayStep : qui permet d'executer pas a pas un scenario.
  Presser le bouton du milieu pour incremeter l'execution.

> Pour passer d'une action a l'autre ou arreter une action :
   Cliquer le bouton de gauche ou de droite.
```

```
> Pour choisir un nouveau nom de fichier pour le scenario :
   Cliquer dans le rectangle du haut, et entrer le nom au clavier.

/*-----------------------------------------------------------------*/
/*-----------------------------------------------------------------*/
MakeXEditScenario(x0, y0, w, h, color, filename)
int x0, y0, w, h, color;
char *filename;

 ouvrent une fenetre en (x0, y0) de taille (w,h) (il faut w>70 et
h>60) avec un boutton de couleur color pour editer et executer un
scenario a partir du fichier de nom filename.
/*-----------------------------------------------------------------*/
```

3.5.5 Processus sans fenêtre : MakeXProcess()

Cette fonction ouvre une pseudo-fenêtre qui ne contient aucun mécanisme graphique, ne reçoit aucun événement lié à la souris, et n'a aucune fenêtre X associée. Une quelconque tentative d'accéder à ces fonctions produit une erreur.

Une telle fenêtre est utilisée pour simuler du multi-processing, elle reçoit les événements (**Starting**), (**Sampling**) et (**Message**(nn)). Elle permet de généraliser les fonctions définies ici, à une fenêtre non-graphique.

```
/*-----------------------------------------------------------------*/
ActiveXWindow *MakeXProcess(SamplingPeriod, EventTask)
int SamplingPeriod;             /* Periode d'echantillonage en
                                   milli-secondes, elle sera approximee
                                   par un multiple de la periode
                                   d'horloge du programme (50msec) */
void (*EventTask) ();           /* Tache de gestion des evenements */
/*-----------------------------------------------------------------*/
```

3.5.6 Moniteur Video Interactif : MakeXVideoMoniteur()

Il est possible de disposer d'un outil d'affichage d'images préprogrammé, qui affiche une image, et permet de lire les valeurs des pixels.

```
/*-----------------------------------------------------------------*/
MakeXVideoMoniteur(x0, y0, xsize0, ysize0, UserGraphTask)
int x0, y0, xsize0, ysize0;
void (*UserGraphTask) ();

 Ouvre en (x0,y0) une fenetre permettant d'afficher des images de
taille au maximum (xsize0, ysize0), la fenetre cpmplete aura une
taille de (xsize0, ysize0+30) pixels. Une seule fenetre de ce type
peut etre ouverte a la fois.

affima(NomImage, im, xsize, ysize, signed)
char NomImage[];
[unsigned] char im[xsize*ysize];
int xsize, ysize;
int signed;

 Affiche une image de nom NomImage, contenue dans le buffer im[].
```

```
  signed vaut 1 si l'image est signee (buffer de char), ou 0 si
l'image est non signee (buffer de unsigned char).

affima_f(NomImage, im, xsize, ysize)
char NomImage[];
float im[xsize*ysize];
int xsize, ysize;

 Affiche une image de nom NomImage, contenue dans le buffer im[].
L'ajustement des niveaux de gris est automatique.

affcourbe(NomCourbe, x, nechs, xmin, xmax)
char NomCourbe[];
double x[], xmin, xmax;
int nechs;

 Affiche une courbe de nom NomCourbe, contenue dans le buffers
(x[nechs]). Le trace va de (xmin,ymax) a (xmax,ymin).

/*-----------------------------------------------------------------*/
```

3.6 Les routines de gestion des fenêtres

3.6.1 Changement et copie de plan de vue : AwUseView(), AwCopyView(

```
/*-----------------------------------------------------------------*/
AwUseView(Win, n)
ActiveXWindow *Win;
int n;

 apres l'appel a AwUseView(), la fenetre Win travaille avec le plan de vue
numero n. L'entier n est pris modulo le nombre de vues.

 au depart, la fenetre travaille avec le plan 0.
/*-----------------------------------------------------------------*/
/*-----------------------------------------------------------------*/
AwCopyView(Win, nsource)
ActiveXWindow *Win;
int nsource;

 recopie la totalite des pixels de la vue de numero nsource dans la
vue courante. L'entier est pris modulo le nombre de vues.

 Utile pour sauvegarder des plans graphiques et les recopier tres
rapidement sans transfert de donnees entre client et serveur X.

/*-----------------------------------------------------------------*/
```

3.6.2 Changement de fonte : AwUseFont(),AwUseOneFont()

```
/*-----------------------------------------------------------------*/
AwUseOneFont(Win,font_name)
```

```
ActiveXWindow *Win;
char font_name[];

AwUseFont(font_name)
char font_name[];

 apres l'appel a AwUseFont(), toutes les fenetres travaillent avec la
fonte dont le nom a ete specifie. apres l'appel a AwUseOneFont() la
fenetre specifiee travaille avec le fonte specifiee. Quelques noms de
fontes courantes, ici :

"cour.b.*" *=10, 12, 14, 16, 18, 24
"cour.r.*" *=10, 12, 14, 16, 18, 24
"5x8",
"6x9", "6x10", "6x12", "6x13", "6x13bold",
"7x13", "7x13bold", "7x14",
"8x13", "8x13bold", "8x16", "8x16kana", "8x16romankana",
"9x15", "9x15bold",
"10x20",
"12x24", "12x24kana", "12x24romankana"

/*-------------------------------------------------------------------*/
```

3.6.3 Changement de curseur : AwUseCursor()

```
/*-------------------------------------------------------------------*/
AwUseCursor(Win,cursor_number)
ActiveXWindow *Win;
int cursor_number;

 apres l'appel a AwUseCursor(), la fenetre specifie utilise le curseur
de numero choisi, qui apparait lors de l'introduction dans la fenetre
et disparait a sa sortie.

 Il y a un tres grand nombre de curseurs predefinis, dont la liste
est donnee dans :

 /usr/include/X11/cursorfont.h

 qui associe par a chaque numero de curseur un define avec un nom.
Ce fichier est deja inclus dans votre source a travers activewin.h.

 Voici quelques exemples :

#define XC_arrow 2
#define XC_based_arrow_down 4
#define XC_based_arrow_up 6
#define XC_coffee_mug 28
#define XC_cross 30
#define XC_tcross 130
#define XC_top_left_arrow 132
#define XC_top_left_corner 134
#define XC_top_right_corner 136
```

```
#define XC_umbrella 146
#define XC_xterm 152

/*-------------------------------------------------------------------*/
```

3.6.4 Changement de comportement : AwNewBehavior()

```
/*-------------------------------------------------------------------*/
AwNewBehavior(Win, behavior)
ActiveXWindow *Win;
int behavior;

 apres l'appel a AwNewBehavior(), la fenetre Win possede le
comportement passe en parametre. Le nombre de vues ne peut pas etre
change dynamiquement.
/*-------------------------------------------------------------------*/
```

3.6.5 Execution manuelle d'un événement : AwSendEvent()

```
/*-------------------------------------------------------------------*/
AwSendEvent(Win, event)
ActiveXWindow *Win;
int event;

 execute la routine de gestion de la fenetre, pour l'evenement passe
en argument. L'evenement Message() ne doit pas e^tre envoye' avec cette
routine mais en utilisant AwSendMess().
/*-------------------------------------------------------------------*/
```

3.6.6 Gestion des messages entre fenêtres AwSendMess()

Le numero du message doit etre compris entre 1 et 256, dans cette version, seul les processus et fenetres qui ont utilisé la macro Message(nn) le reçoivent. Chaque message doit avoir un numéro différent et la structure de donnee du C "enum" permet d'assurer une bonne numerotation des messages.

Lors de la reception d'un message, les informations du message sont disponibles dans
`char *Win->buffer;`
qui est un pointeur vers le buffer contenant le message.

Le buffer du message ne doit JAMAIS etre modifie par le recepteur du message, il doit etre seulement lu.

```
/*-------------------------------------------------------------------*/
AwSendMess(Win, CodeMessage, BufferMessage)
ActiveXWindow *Win;
char *BufferMessage;
int CodeMessage;

 envoie le message de code Message, a la fenetre Win. Le contenu du
message est dans le buffer pointe par BufferMessage. La variable
BufferMessage est un pointeur e'ventuel vers une zone de donne'e
contenant le message.

 AwSendMess() envoie le message a toutes les fenetres si Win == 0.
```

```
 La fenetre Win doit avoir une structure de la forme :

WinTask(Win)
ActiveXWindow *Win;
{
  switch (Win->Event) {

   .../...

  case Message(CodeMessage): {

   .../...

  } break;

   .../...

  }
}

/*-------------------------------------------------------------------*/
```

On pourra faire trois remarques :

(1) Ce procédé de messagerie est "de bas niveau", il permet simplement de faire communiquer plusieurs processus, comme des interruptions hardware. Il se prête bien à la notion de système asynchrone réactif à un événement. Il est très rapide à l'exécution. On remarquera que ces messages sont définis une fois pour toute, comme des canaux de communication.

(2) Le gros inconvénient est que le système est fragile : si l'utilisateur [1] déclare n'importe comment les messages, ou [2] écrit dans le buffer du message à la réception alors qu'il ne devrait que lire ce message, tout peut arriver.

(3) En pratique, les choses se passent ainsi, le contrôle est passé successivement à chaque fenêtre ou processus destinataires du message, avec les bons paramètres, dans l'ordre chronologique où ils ont été définis. Le contrôle est rendu à la routine qui a appelé AwSendMess() ensuite.

3.6.7 Dump de fenêtres dans un fichier PostScript : DumpXWin()

```
/*-------------------------------------------------------------------*/
DumpXWin(Win, filename)
ActiveXWindow *Win;
char filename[];

 Stocke la fenetre dans un fichier de nom filename.ps en format
PostScript pour son impression ou son insertion dans un document.
Utilise l'utilitaire XtoPS(). Si Win==0, stocke tout le display.

/*-------------------------------------------------------------------*/
```

3.6.8 Modification de la courbe des niveaux de gris : AwGreyLevel() MakeXGreyLevelButton()

```
/*-------------------------------------------------------------------*/
AwGreyLevel(power)
double power;
```

```
 Modifie la courbe de reponse des niveaux de gris a l'affichage.  Un
affichage agreable des images impose en general d'avoir une courbe
de reponse concave, et l'on utilise ici une courbe de la forme :
       power
  y = x
ou power est plus petit que 1, et vaut 1/2 par defaut. Pour power=1
on a un affichage lineaire.

/*--------------------------------------------------------------------*/
/*--------------------------------------------------------------------*/
MakeXGreyLevelButton(x0,y0,h,v)
int x0,y0,h,v;

 MakeXGreyLevelButton() est un boutton qui permet le reglage de la
courbe de reponse des niveaux de gris a l'affichage.

/*--------------------------------------------------------------------*/
```

3.7 Les routines de tracé graphique

3.7.1 Copie d'image dans une fenêtre : DisplayXImage(), GetXImage(), GetXImage()

```
/*--------------------------------------------------------------------*/
DisplayXImage(Win, x0, y0, Im, xdim, ydim)
unsigned char Im[xdim * ydim];
int xdim, ydim;
ActiveXWindow *Win;
int x0, y0;

 recopie une image de taille (xdim, ydim) dans le buffer Im[], l'image
est affichee dans le fenetre Win, a partir de la position (x0, y0) dans
la fenetre. L'utilisation de CodeImageBuffer() n'est plus necessaire.

/*--------------------------------------------------------------------*/
/*--------------------------------------------------------------------*/
GetXImage(Win, x0, y0, Im, xdim, ydim)
GetXWin(Win, x0, y0, Im, xdim, ydim)
unsigned char Im[xdim * ydim];
int xdim, ydim;
ActiveXWindow *Win;
int x0, y0;

 recopie une image de taille (xdim, ydim) dans le buffer Im[], l'image
est prelevee dans la fenetre Win, a partir de la position (x0, y0) dans
la fenetre.

 GetXImage() transforme les couleurs et les niveaux de gris pour que
le re'sultat puisse etre utilisee comme une image a 256 niveaux de gris,
GetXWin() laisse les niveaux de gris inchanges.

 Cette fonction altere les images sun une station a deux niveaux de
gris.
```

```
/*-------------------------------------------------------------------*/
```

3.7.2 Tracé de points et courbes dans une fenêtre : DrawXPoint() DrawXCross() DrawXCourbe()

```
/*-------------------------------------------------------------------*/
DrawXPoint(Win, x1, y1, color)
DrawXCross(Win, x1, y1, color)
ActiveXWindow *Win;
int x1, y1;
unsigned char color;

 Trace un point (x1, y1) avec la couleur color dans la fenetre Win.
DrawXPoint() colorie un seul point de l'ecran, DrawXCross() une petite
croix de 2x2 pixels.
/*-------------------------------------------------------------------*/
/*-------------------------------------------------------------------*/
DrawXCourbe(Win, Pts, NbPts, Color, Mode=0)
ActiveXWindow *Win;
XPoint Pts[];
int NbPts, Color, Mode;

 trace une courbe de NbPts points avec la couleur Color dans la fenetre
Win.
 Le parametre Mode doit valoir 0 dans cette version et est prevu pour
les futures extensions.
 Le tableau de points Pts[] utilise la structure X-window :

typedef struct {
    short x, y;
} XPoint;

pour definit un point.
/*-------------------------------------------------------------------*/
```

3.7.3 Tracé d'une ligne dans une fenêtre : DrawXLine(), DrawXTrait(),DrawXVect()

```
/*-------------------------------------------------------------------*/
DrawXLine(Win, x1, y1, x2, y2, color)
DrawXTrait(Win, x1, y1, x2, y2, color, width)
DrawXVect(Win, x1, y1, x2, y2, color)
ActiveXWindow *Win;
int x1, y1, x2, y2;
unsigned char color, width;

 DrawXLine trace une ligne du point (x1, y1) au point (x2, y2) avec la
couleur color dans la fenetre Win.
 DrawXTrait trace une ligne du point (x1, y1) au point (x2, y2) avec la
couleur color et l'epaisseur de trait width dans la fenetre Win.
 DrawXVect trace une ligne du point (x1, y1) au point (x2, y2) avec la
couleur color dans la fenetre Win, une fleche est placee a l'extremite
(x2, y2) du segment.
```

```
 La pseudo-couleur InvVideo permet d'afficher la ligne par inversion
video.
/*-------------------------------------------------------------------*/
```

3.7.4 Tracé d'un rectangle dans une fenêtre : DrawXRectangle(), FillXRectangle()

```
/*-------------------------------------------------------------------*/
DrawXRectangle(Win, x1, y1, w, h, color)
FillXRectangle(Win, x1, y1, w, h, color)
ActiveXWindow *Win;
int x1, y1, w, h;
unsigned char color;

 trace un rectangle de coin superieur gauche (x1, y1), de largeur w
(width), de hauteur h (height) et de couleur color dans la fenetre Win.

 le dernier point trace a les coordonnes (x1 + w - 1, y1 + h - 1).
/*-------------------------------------------------------------------*/
```

3.7.5 Tracé d'une ellipse oblique : DrawXEllipse()

```
/*-------------------------------------------------------------------*/
DrawXEllipse(Win, x0, y0, alpha, lmax, lmin, color)
ActiveXWindow *Win;
int x0, y0, lmax, lmin;
double alpha;
unsigned char color;

 trace une ellipse de centre (x0,y0), dont le grand axe de longueur
lmax est incline de alpha, lmin etant la longueur du petit axe. Le
trace se fait avec la couleur color dans la fenetre Win.

 x0, y0, lmax, lmin sont donnes en pixel.
 alpha              est  donne en  radian.
/*-------------------------------------------------------------------*/
```

3.7.6 Tracé de caractères dans une fenêtre : DrawXString()

```
/*-------------------------------------------------------------------*/
DrawXString(Win, x1, y1, string, color)
ActiveXWindow *Win;
int x1, y1;
char string[];
unsigned char color;

 trace une chaine de caracteres centree au point (x1, y1) avec la
couleur color dans la fenetre Win.
/*-------------------------------------------------------------------*/
```

3.7.7 Taille d'une chaîne de caractères : SizeXString()

```
/*-------------------------------------------------------------------*/
```

```
SizeXString(Win, string, width, height)
ActiveXWindow *Win;
char string[];
int *width, *height;

 renvoie la largeur et la hauteur en pixels d'une chaine de caracteres
correspondant a sa taille si elle etait affichee dans la fenetre.
/*-----------------------------------------------------------------*/
```

3.7.8 Effacement d'une fenêtre : ClearXWindow() InvertXColors()

```
/*-----------------------------------------------------------------*/
ClearXWindow(Win, Color)
ActiveXWindow *Win;
unsigned char Color;

 efface la fenetre et met le fond a une couleur uniforme.
/*-----------------------------------------------------------------*/
/*-----------------------------------------------------------------*/
InvertXColors(Win)
ActiveXWindow *Win;

 inverse les couleurs de tous les pixels de la fenetre. Produit
un effet de flash.
/*-----------------------------------------------------------------*/
```

3.8 Gestion de la fenêtre en terminal

Il est possible d'utiliser les fenetres commes des terminaux alphanumériques et d'y faire des entrées sorties de caractères. Le fond est alors blanc et le texte en noir. On utilise une syntaxe similaire aux printf et scanf du C.

3.8.1 Sortie de caractères : Awprintf(), Awputchar(), AwMovePrintf()

```
/*-----------------------------------------------------------------*/
Awprintf(Win, format, [args ...])
ActiveXWindow *Win;
char *format;

 Effectue un printf dans la fenetre Win. Identique en tous points
au printf du C. Les caracteres de controle sont normaux et les
effets suivants :

 \n    : passage a la ligne (RETURN+LINE FIELD).
 \r    : revient en debut de ligne sans passer de ligne (superposition).
 \t    : ecrit 6 blancs.
 \014  : efface l'ecran, revient en debut de fenetre (clear).
 \010,
 \177  : revient d'un caractere et efface (Back-Space) et (Delete).

Awputchar(Win, c)
ActiveXWindow *Win;
char c;
```

```
 Sort un caractere dans la fenetre indiquee, comme ci-dessus.

AwMovePrintf(Win, x, y)
ActiveXWindow *Win;
int x, y;

 Positionne la sortie de caracteres a partir de la position (x,y) de
la fenetre. Apres cet appel, la gestion de la position du trace est
geree de nouveau automatiquement.

/*-----------------------------------------------------------------*/
```

3.8.2 Entrées de caractères : Awscanf(), Awgetchar(), Awreadchar(), Awnochar(), Awkbhit()

```
/*-----------------------------------------------------------------*/
Awscanf(Win, format, [args ...])
ActiveXWindow *Win;
char *format;

 Effectue un scanf dans la fenetre Win. Identique en tous points
au scanf du C.

char Awgetchar(Win)
char Awreadchar(Win)
ActiveXWindow *Win;

 Lit un caractere dans la fenetre indiquee, comme ci-dessus. Cette
lecture est bloquante. Awgetchar() ecrit le caratere entre au clavier,
Awreadchar() ne l'ecrit pas.

Awnochar(Win)
ActiveXWindow *Win;

 Elimine toud les caracteres precedemments entres au clavier, le premier
caractere lu sera celui entre apres l'appel a cette routine.

int Awkbhit(Win)
ActiveXWindow *Win;

 Renvoie 1 si il y a un caractere a lire, 0 sinon. Permet de tester
l'occurence de caractere en entree. Ils sont ensuite relus avec
Awgetchar().

/*-----------------------------------------------------------------*/
```

3.8.3 Entrées de paramètres : AwLitInt(), AwLitDouble(), AwLitCh(), AwOuiNon(), AwLitSt()

Routines de dialogue identiques à celles proposéees dans l'annexe, section B.2, permet de simplifier la lecture de paramètres.

```
/*-----------------------------------------------------------------*/
```

```
int     AwLitInt(Win, Question)
double  AwLitDouble(Win, Question)
char    AwLitCh(Win, Question)
int     AwOuiNon(Win, Question)
        AwLitSt(Win, Question,Reponse)
ActiveXWindow *Win;
char *Question, Reponse[];

/*-----------------------------------------------------------------*/
```

3.9 Gestion de segments rémanents

Voici les routines qui permettent de gérer les segments rémanents :

3.9.1 Création d'un segment rémanent : CreateXSegment()

```
/*-----------------------------------------------------------------*/
CreateXSegment(Win, x1, y1, x2, y2, segmentindex, color)
ActiveXWindow *Win;
int x1, y1, x2, y2, segmentindex, color;

 Cree un segment remament d'extremites (x1, y1) <-> (x2, y2) attache a
la vue courante de la fenetre (comme selectionnee par AwUseView()).
Un index pour referencer le segment, il doit etre unique.

 Le segment est ensuite affiche avec la couleur desiree.

 L'index passe en argument est ensuite utilise pour identifie le
segment lors de sa selection par le comportement (SelectSegment).

/*-----------------------------------------------------------------*/
```

3.9.2 Clignotement d'un segment : BlinkXSegment(), DeleteAllSegments()

```
/*-----------------------------------------------------------------*/
BlinkXSegment(Win, segmentindex)
ActiveXWindow *Win;
int segmentindex;

 Inverse la couleur du segment dont l'index a ete donne. Il faut appeler
deux fois BlinkXSegment() pour retrouver la situation initiale.

/*-----------------------------------------------------------------*/
```

3.9.3 Destruction d'un segment rémanent : DeleteXSegment(), DeleteAllSegments()

```
/*-----------------------------------------------------------------*/
DeleteXSegment(Win, segmentindex)
ActiveXWindow *Win;
int segmentindex;

 Enleve le segment de la liste des segments et tente de l'effacer a
```

```
l'ecran.

/*--------------------------------------------------------------------*/
/*--------------------------------------------------------------------*/
DeleteAllSegments(Win)
ActiveXWindow *Win;

 Enleve tous les segments et tente des les effacer de l'ecran.

/*--------------------------------------------------------------------*/
```

3.10 Simulation d'un oscilloscope numérique dans une fenêtre.

Il est possible de gérer une ou des trace d'oscilloscope dans une fenêtre. Au fur et à mesure de leur envoi à la fenêtre les écchantillons sont affichés et la trace décalée automatiquement. La surface graphique sur laquelle est affichée la trace est sauvegardée et restaurée au fur et à mesure. Une fenêtre active peut gérer jusqu'à 8 traces de scope;

3.10.1 Définition d'une trace d'affichage : InitVoieScope(), CloseVoieScope()

```
/*--------------------------------------------------------------------*/
InitVoieScope(Win, Voie, Xmin, Xmax, Mode)
ActiveXWindow *Win;
int Voie;
double Xmin, Xmax;
int Mode;

 Definit une trace d'affichage dans la fenetre Win, de numero Voie.
Le numero de Voie peut aller de 0 a 7. C'est a l'utilisateur de
choisir un numero de voie.

 Les echantillons prennent leur valeur entre Xmin et Xmax, une valeur
Xmin sera affichee en bas de la fenetre et une valeur Xmax en haut de la
fenetre.

 La trace se fait par inversion de couleur. Un certain nombre de modes
d'affichage de la trace sont possibles, le plus simple etant de mettre
Mode a zero :

 La trace peut soit etre affichee en tournant (ScopeRollingMode), soit
en decalant l'ecran a chaque fois (ScopeScrollingMode) :

#define ScopeRollingMode          0x0000
#define ScopeScrollingMode        0x0001

 Les points peuvent formes une courbe (DotPlottingMode) ou des barres
verticales d'extremite le milieu de la trace (MiddleBarPlottingMode) ou
le bas de la trace (DownBarPlottingMode) :

#define DotPlottingMode           0x0000
#define MiddleBarPlottingMode     0x0010
#define DownBarPlottingMode       0x0020
```

```
 L'echelle de la trace peut etre ajustee automatiquement, elle peut
etre recentree (AutoCenter) et/ou ajustee en amplitude (AutoScale).

#define AutoCenter                  0x0100
#define AutoScale                   0x0200

/*--------------------------------------------------------------------*/
/*--------------------------------------------------------------------*/
CloseVoieScope(Win, Voie)
ActiveXWindow *Win;
int Voie;

  detruit la trace actuellement affichee et remet l'ecran en etat, il
faut reutiliser InitVoieScope() pour un nouvel affichage.

/*--------------------------------------------------------------------*/
```

3.10.2 Affichage d'un echantillon : OutScope()

```
/*--------------------------------------------------------------------*/
OutScope(Win,Voie,Valeur)
ActiveXWindow *Win;
int Voie;
double Valeur;

 Affiche l'echantillon de valeur Valeur dans la trace Voie de la fenetre
Win, la Voie devra avoir ete prealablement delaree.
/*--------------------------------------------------------------------*/
```

3.11 Création et utilisation de scénarios

Il est possible de mémoriser et de réexécuter une séquence d'actions de l'utilisateur (manipulation de la souris et du clavier). La gestion des événements mémorisés est entièrement automatique.

3.11.1 Mémorisation d'un scénario : SaveScenario(), CloseScenario()

```
/*--------------------------------------------------------------------*/
int SaveScenario(filename)
char filename[];

CloseScenario()

 L'appel a SaveScenario() demarre le stockage de toutes les commandes
souris et clavier entrees par l'utilisateur dans le fichier de nom
filename. SaveScenario() renvoie 1 si le fichier a ete ouvert 0 si
on ne peut stocker dans ce fichier.

 L'appel a CloseScenario() termine le stockage de toutes les commandes
souris et clavier entrees par l'utilisateur, le fichier est ferme.

 Le fichier est un fichier ASCII, a raison d'une commande par ligne,
au format suivant :
```

```
Win ''Win->index'' ''Event'' 'char' souris=(''x'', ''y'')

qui memorise donc l'index de la fenetre, le caractere entre au clavier,
et la position de la souris dans le fenetre.

 Ce type de fichier peut etre manipuler avec les outils Unix (editeur,
grep, sed, sort, etc...).

/*-------------------------------------------------------------------*/
```

3.11.2 Exécution d'un scénario : StartScenario(), NextEventScenario()

```
/*-------------------------------------------------------------------*/
int StartScenario(filename)
char filename[];

 Demarre l'execution d'un scenario stocke dans le fichier de nom
filename. La routine renvoie 1 si le fichier existe, 0 sinon. Les
differentes commandes du scenario sont executes sequenciellement
par la commande :

int NextEventScenario()

 qui execute la premiere commande du fichier de scenarion, et avance
sequenciellement dans ce fichier.  La routine renvoie 1 si une commande
a ete effectivement lue, et 0 si le fichier etait termine.

/*-------------------------------------------------------------------*/
```

3.12 Transformations de coordonnées 2D-3D

3.12.1 Définition des coordonnées 1D : AwSet1Dcoord()

On appelle coordonnées 1D des changements d'echelles sur chaque axe (horizontal et vertical) sans rotations de la figure. On utilisera des macros de façon à pouvoir être utilisé avec toutes les routines graphiques, sans avoir à définir de routines supplémentaires. Il faut se souvenir que les tracés se font par défaut avec l'echelle 1 unité = 1 pixel écran.

```
/*-------------------------------------------------------------------*/
AwSet1Dcoord(Win, Xmin, Xmax, Ymin, Ymax)
ActiveXWindow *Win;
double Xmin, Xmax, Ymin, Ymax;

 Specifie les coordonnees pour un changement d'echelle dans la
fenetre. Le trace se fera de Xmin (correspondant au pixel 0) a Xmax
(correspondant au dernier pixel) en horizontal, et identiquement de
Ymin a Ymax en vertical. Les transformations de coordonnees se font
avec x1D() et y1D().
/*-------------------------------------------------------------------*/
```

3.12.2 Utilisations des coordonnées 1D : x1D() y1D()

```
/*-------------------------------------------------------------------*/
int x1D(Win, X)
```

```
int y1D(Win, Y)
ActiveXWindow *Win;
double X, Y;

 Transforme les coordonnees flottantes en pixel, ceci en fonction des
coordonnees 1D specifiees par AwSet1Dcoord().

 On utilise typiquement ces macros lors de passage d'arguments :

      AwSet1Dcoord(Win, Xmin, Xmax, Ymin, Ymax);

      .../...
      double X1, Y1, X2, Y2;

      DrawXLine(Win,
       x1D(Win, X1), y1D(Win, Y1), x1D(Win, X2), y1D(Win, Y2), color);

cette methode evite de redefinir les routines de trace pour chaque
type de coordonnees.
/*-------------------------------------------------------------------*/
```

3.12.3 Définitions des coordonnées 3D : AwSet3Dcoord()

```
/*-------------------------------------------------------------------*/
AwSet3Dcoord(Win, R, t)
ActiveXWindow *Win;
double R[3][3], t[3];


 Specifie les coordonnees pour un changement de repere est une
projection.  Les transformations de coordonnees se font
avec x3D() et y3D().
/*-------------------------------------------------------------------*/
```

Un point $M_1 = (X_1, Y_1, Z_1)$ subit une transformation affine (rotation, changement d'échelle) de matrice R puis une translation de vecteur t, et enfin une projection perspective. Le point image $m = (x, y)$ exprimé en pixel correspond donc au point $M1$ par les équations :

$$\begin{vmatrix} X \\ Y \\ Z \end{vmatrix} = R \cdot \begin{vmatrix} X_1 \\ Y_1 \\ Z_1 \end{vmatrix} + \mathrm{t} \ \textit{and} \begin{cases} x = X/Z \\ y = Y/Z \end{cases}$$

3.12.4 Utilisations des coordonnées 3D : x3D() y3D()

```
/*-------------------------------------------------------------------*/
int x3D(Win, X, Y, Z)
int y3D(Win, X, Y, Z)
ActiveXWindow *Win;
double X, Y, Z;

 Transforme les coordonnees 3D flottantes en pixel 2D, ceci en
fonction des coordonnees specifiees par AwSet3Dcoord().

 On utilise typiquement ces macros lors de passage d'arguments :
```

```
      AwSet3Dcoord(Win, R, t);

      .../...
      double X1, Y1, Z1, X2, Y2, Z2;

      DrawXLine(Win, x3D(Win, X1, Y1, Z1), y3D(Win, X1, Y1, Z1),
        x3D(Win, X1, Y1, Z1), y3D(Win, X2, Y2, Z2), color);

cette methode evite de redefinir les routines de trace pour chaque
type de coordonnees.
/*-------------------------------------------------------------------*/
```

4 Liaison avec la programmation de la Xlib et des Widgets

4.1 La structure de donnée ActiveXWindow

Voici les champs de cette structure :

```
typedef struct {

/*********************************************************************/
/***            CHAMPS POUR L'INTERFACE AVEC LA XLIB                 ***/
/*********************************************************************/

  Window win;     /* Index de la fenetre X attachee a la fenetre active */
  Display *display; /* Pointeur vers le display ou se fait le graphique */
  int screen;       /* Index de l'ecran ou est affiche le graphique */
  Visual *vis;      /* Pointeur vers le visuel d'affichage */
  GC gc;            /* Contexte graphique actuel de la fenetre,
                       il est modifie au cours du temps */
  int depth;
  int width;
  int height;
  int bgcolor;      /* Profondeur, Largeur, Hauteur et Couleur de fond
                       de la fenetre utilisee */

  int Event;        /* Dernier evenement recu par la fenetre */

/*********************************************************************/
/***      PARAMETRES UTILISES POUR LA GESTION DES EVENEMENTS         ***/
/*********************************************************************/

 int index ; /* Numero de la fenetre.  C'est un numero de 1 a n,
ou n est le nombre de fenetres.  Il permet de specifier de maniere
unique la fenetre.  Il correspond a l'ordre dans lequel les
fenetres ont ete allouees */

 int localtime; /* Nombre d'echantillonnages depuis le debut de
l'existence de la fenetre, compte depuis 0.  Le temps presque reel
est donc la periode d'echantillonnage de la fenetre, multipliee par
cet entier.  */
```

```
 int x, y; /* Position de la souris dans la fenetre.  C'est aussi la
position du reticule dans le cas du comportement (ReadReticule), la
position du coin superieur gauche du rectangle dans le cas du
comportement (ReadRectangle) et la position de l'extremite superieure
gauche du segment dans le cas du comportement (ReadLine) ou
{ReadSemiLine). */

 int x1, y1; /* Position du coin inferieur droit du rectangle dans le
cas du comportement (ReadRectangle) ou Position de l'extremite
inferieure droite du segment dans le cas du comportement (ReadLine) ou
{ReadSemiLine). */

 int i; /* Index du segment actif selectionne par le comportement
(SelectSegment). Il vaut 0 si aucun segment n'a ete selectionne */

 double a; /* Parametre lu par une fenetre predefinie de type curseur
ou volant */

 int Button1Down, Button2Down, Button3Down; /* Vaut 1 si le bouton de
gauche(1), du milieu(2), ou de droite(3) a ete enfonce, 0 sinon */

 char *buffer; /* Pointe vers le buffer contenant un message */

.../...

} ActiveXWindow;
```

4.2 Appels à des routines de la Xlib

On a décrit comment utiliser les fenêtres actives en fait sans avoir à se préocuper des appels au serveur X. En réalité il est très facile d'utiliser la Xlib pour chacune des fenêtres, sachant que la structure de donnée attachée à une fenêtre contient les informations nécessaires.

Le package décrit ici s'interface donc de manière évidente avec les appels de la Xlib. Par exemple pour appeler la routine qui permet de passer en inverse vidéo, il faut taper :

```
ActiveXWindow *Win;

 .../...

  XSetFunction(Win->display, Win->gc,
   GXinvert);

  XFillRectangle(Win->display, Win->win, Win->gc,
   0, 0, Win->w - 1, Win->h - 1);

  XSetFunction(Win->display, Win->gc,
   GXcopy);
```

On remarquera que tous les paramètres de service sont présent dans la structure ActiveXWindow.

Néanmoins les appels à la Xlib ne sont alors plus gérés de manière transparente par le noyau des fenêtres actives, et il faut reprendre manuellement la gestion des fenêtres.

4.3 Utilisation conjointe avec des widgets

Les fenêtres actives *sont des widgets*. Ils sont compatibles avec les widgets *intrinsic* de la Xlib, et d'autres widgets.

Non seulement elles sont compatibles, mais elles ont été rendues complètement asynchrones avec le reste du programme, de façon à ne pas à avoir à gérer conjointement les deux mécanismes.

Une seule précaution : il faut attacher à la fenêtre principale un widget et préciser que les événements sont gérés par la fenêtre active. Les processus permettant ce mode de fonctionnement sont les suivants :

- SetActiveWindow() qui definit un widget dont la fenetre contiendra le display de toutes les fenetres actives qui seront regroupes dans une seule surface rectangulaire. Cette routine a été définie de la manière suivante, pour les widgets Athena :

```
/*--------------------------------------------------------------------*/
void SetActiveWindow(widget, parentwidget, width, height)
Widget *widget, parentwidget;
int width, height;
{
  Arg args[2];
  int num_args;

  XtSetArg(args[0], XtNwidth, (Dimension) width);
  XtSetArg(args[1], XtNheight, (Dimension) height);
  num_args = 2;
  *widget = XtCreateManagedWidget("ActiveWinWidget", simpleWidgetClass,
   parentwidget, args, num_args);
}
/*--------------------------------------------------------------------*/
```

- InitXDisplayWithWidget() qui initialise les fenetres actives, se met à la place de InitXDisplay() et prépare le noyau des fenêtres de manière compatible avec les widgets.
- AWMainLoop() qui remplace XtMainLoop() ou GereActiveXWindows() et qui dispatche les événements vers les deux mécanismes. Cette routine a été définie de la manière suivante, et fonctionne pour les widgets Athena et Motif :

```
/*--------------------------------------------------------------------*/
AWMainLoop()
{
  XEvent event;

  StartActiveXWindows();
  while (1) {
   ActiveWinXNextEvent(&event);
   XtDispatchEvent(&event);
 }
}
/*--------------------------------------------------------------------*/
```

4.3.1 Gestion désynchronisée des fenêtres actives en présence de widgets : SetActiveWindow(), InitXDisplayWithWidget(), AWMainLoop()

Voici les paramètres d'appel de ces routines :

```
/*--------------------------------------------------------------------*/
void SetActiveWindow(widget, parentwidget, width, height)
/* Definit un widget qui contiendra une fenetre active */
Widget *widget, parentwidget;
```

```
int width, height;

 genere un widget reference par widget de parent parentwidget et
de taille width x height.

InitXDisplayWithWidget(dpy, parentwindow, toplevelwindow)
/* Initialise le mecanisme des fenetres actives */
Display *dpy;
Window parentwindow, toplevelwindow;

 initialise les fenetres actives pour le display dpy. La fenetre
parentwindow est la fenetres qui contiendra toutes les fenetres
actives, la fenetre toplevelwindow est la fenetre principale de
l'application, fille de la root window, et a laquelle on affectera la
color-map.


AWMainLoop()
/* Routine de gestion de la boucle principale du programme */

 dispatche les evenements vers les differentes fenetres.

/*-----------------------------------------------------------------------*/
```
Un exemple de structure de programme est le suivant :
```
/*-----------------------------------------------------------------------*/

main(argc, argv)
int argc;
char **argv;
{
  XtAppContext app_con;
  Widget widget_aw;

  toplevel = XtAppInitialize( ...
  box = XtCreateManagedWidget( ...

  .../..

  /* Voila la declaration du widget contenant le display de fenetres
     actives du programme */
  SetActiveWindow(&widget_aw, , 512, 512);

  .../...

 /* Voila l'initialisation des fenetres actives pour fonctionner
    avec les widgets, remplace InitXDisplay() */
 InitXActiveWithWidget(
  XtDisplay(toplevel),
  XtWindow(widget_aw),
  XtWindow(toplevel));

 MakeXWindow(...
```

```
 /* Voila la ou on lance la gestion conjointe des deux mecanismes */
 AWMainLoop(app_con);
}
/*-------------------------------------------------------------------*/
```

4.3.2 Declaration du widget pour la bibliothèque Motif

Dans le cas de la bibliothèque Motif, il faut remplacer SetActiveWindow() par la routine suivante qui n'est pas en bibliothèque, mais doit être incluse par l'utilisateur.

```
/*-------------------------------------------------------------------*/

SetActiveWindowMotif(widget, parentwidget, width, height)
Widget *widget, parentwidget;
int width, height;
{
  Arg args[2];
  Widget awW ;
  int num_args;

  XtSetArg(args[0], XmNwidth, (Dimension) width);
  XtSetArg(args[1], XmNheight, (Dimension) height);
  num_args = 2;
  awW = XmCreateForm (parentwidget, "awW", args, num_args);
  XtManageChild (awW) ;
  *widget = XmCreateDrawingArea(awW, "ActiveWinWidget", args, num_args);
  XtManageChild (*widget) ;
}
/*-------------------------------------------------------------------*/
```

Tout le reste de la programation reste identique.

A Compilation et Edition de lien ...

Le fichier commencera par :

```
#include <stdio.h>
#include <math.h>
#include "activewin.h"

main( ...
```

où a été défini dans activewin.h tous les includes de la Xlib.
La compilation se fera avec les options suivantes pour pouvoir accéder aux includes et aux librairies de routine.

```
# Generation des fichiers objet .o
cc -c $file.c -I/usr/include/X11R4 -I/u/corse/1/thierry/src /include

# Edition de lien
cc -o $HOME/bin/$program_name $objets \
 /u/corse/1/thierry/bin/thglib.a /usr/lib/X11R4/libX11.a -lm
```

Si l'acces a corse n'est pas possible utiliser les copies de thlib.a et activewin.h dans :

```
 /net/lib/maple.local/lib/mpls/sun[34]
```

A.1 Quelques scripts pour la mise au point de programmes de fenêtres actives sur un Sun3/4

A.1.1 Compilation et exécution automatique de programmes : ccc

La shell script ccc a été conçue pour faciliter la compilation et l'exécution de programmes de petite et moyenne importances. Les bibliothèques d'usage courant (*lnag*, *inrimage*, *Xlib*, *pixrect*) sont déclarées. Pendant la mise au point (option -g), elle lance les programmes sous gdb.
En voici l'usage :

```
Usage : ccc [-r|-m|-o] [-g] file_name(s)[.c] ...[: program arguments]
 ccc compiles and links a C-source file using standart library,
  and stores the result in a bin-directory
 If the compilation is successfull the program is run,
  with the given arguments, after confirmation.
 The program name is the name of the first file given
Options :
 -r[un]    : The program is automaticaly runned after compilation
 -o[bj]    : The program is not runned but only compiled
 -g[db]    : The program is automaticaly runned under gdb (debug)
 -m[ake]   : The compilation is done running make,
             instead of compiling the source file at each call
 -c[lear]  : The screen is cleared before compiling
: args     : The rest of the line is taken as arguments to the
              final program
```

A.1.2 Remise en forme de fichiers source en C : cf

Il est possible de remettre en page de manière automatique un fichier de source C d'extension .c ou .h.
La mise en page respecte le standard Kernighan et Ritchie, et facilite la relecture de vos programmes. La synatxe peut être vérifiée avec l'option -c.
On utilise les programmes indent et lint.

En voici l'usage :

```
Usage: cf [-c] [file.c|file.h] ...
 cf beautify, check, and reformat C source file(s)
 indent and lint are used with predefined options
Option :
 -c : lint is applied on the file, to check the syntax
```

Attention ! : Les expressions du type :

```
 int pt = &val, val = *pt, val = -x + y;
```

sont transformées en :

```
 int pt &= val, val *= pt, val -= x + y;
```

selon une règle de syntaxe destinée à éviter les anachronismes. Il faut donc à tout prix parenthéser ce type d'expressions.

```
 int pt = (&val), val = (*pt), val = (-x + y);
```

A.2 Où se trouvent les libs, includes et sources

A.2.1 Accès pour les membres du groupe robotvis

```
# La librairie des routines
LIBDIR=/u/corse/1/thierry/bin/
LIB=$LIBDIR/thlib.a          # Compilee avec l'option -O (optimisee)
LIB=$LIBDIR/thglib.a         # Compilee avec l'option -g (pour debug)

# Le fichier des includes
INCDIR=/u/corse/1/thierry/src/include
INC=$INCDIR/activewin.h

# Les fichier sources
SRCDIR=/u/corse/1/thierry/src /X11/activewin
SRC="$SRCDIR/activewin.c"

# La doc
DOCDIR=/u/corse/1/thierry/src /X11/activewin
DOC=$DOCDIR/activewin.dvi

# Les shells
SHELLDIR=/u/corse/1/thierry/shell
SHELLS="$SHELLDIR/ccc $SHELLDIR/cf"
```

A.2.2 Acces pour les autres utilisateurs de l'INRIA

Le package est, pour des raisons historiques, avec les outils Maple.

```
# La librairie des routines
LIBDIR=/net/lib/maple.local/lib/mpls/sun[34]
LIB=$LIBDIR/thlib.a          # Compilee avec l'option -O (optimisee)

# Le fichier des includes
INCDIR=/net/lib/maple.local/lib/mpls/
INC=$INCDIR/activewin.h

# La doc
DOCDIR=/net/lib/maple.local/lib/mpls/
```

B Quelques autres routines pour les programmes de robotique et vision

B.1 Lecture de paramètres du programme

On propose ici une méthode et des routines pour relire des paramètres d'un programme. Très classiques (voir inrimage, C-microsoft), ces routines permettent d'éditer les arguments du programme en toute facilité.

Une méthode pour relire les paramètres d'un programme Voici comment procéder :

```
/* 1) Il faut declarer ici des variables globales qui vont pointer
      vers le argc et argv. */

int xargc;
char **xargv;

/* 2) Il faut declarer le main et ses arguments et les recopier dans
      xargc et xargv. */

main(argc, argv)
int argc;
char *argv[];
{
  xargc = argc;
  xargv = argv;

/* 3) On lit les parametres du programme avant toute chose */
  {

/* 3.1) On lit toutes les options avec ou sans parametre */

    AvecTrace = IsOption("-t");          /* Exemple */
    AvecVisu = IsOption("-v");           /* Exemple */

    if (!LitOptionInt("-x", &x))         /* Exemple */
        x=valeur_de_x_par_defaut;

/* 3.2) On regarde si il reste encore des options. Si oui on proteste. */

    if (!YaPlusAucuneOption()) {
        fprintf(stderr, "usage: programme arguments\n");
        exit(-1);
    }

/* 3.3) On lit les fichiers passes en parametre. */
    NbFichiers = 0;
    while (LitOptionNomFichier(NomFichiers[NbFichiers]))
     NbFichiers++;

/*       note : on suppose avoir declare quelques part un truc du genre :
```

```
int NbFichiers;
char NomFichiers[NbMaxDeFichiers][256];
*/
  }

.../..
}
```

Voici les routines qui permettent de lire les paramètres d'un programme :

B.1.1 Test de l'occurence d'une option : IsOption()

```
/*-----------------------------------------------------------------*/
 int IsOption(Option)
 char Option[];

  renvoie 1 si l'option de nom Option est presente 0 sinon.  Par
convention une option est de la forme ''-option''. Toutes les options
doivent commencer par un ''-''. Apres avoir ete lue, l'option est
retiree de la liste d'arguments.
/*-----------------------------------------------------------------*/
```

B.1.2 Lecture d'une option à paramètres : LitOptionString(), LitOptionInt(), LitOptionDouble()

```
/*-----------------------------------------------------------------*/
 int LitOptionString(Option, String)
 char Option[];
 char String[];

  renvoie 1 si l'option de nom Option est presente 0 sinon. Si oui,
lit l'argument suivant comme un string et le renvoie. La string ne
doit pas forcement commencer par un ''-''. Apres avoir ete lus,
l'option et le parametre sont retires de la liste d'arguments.

/*-----------------------------------------------------------------*/
 int LitOptionInt(Option, Int)
 char Option[];
 int  *Int;

  renvoie 1 si l'option de nom Option est presente 0 sinon. Si oui,
lit l'argument suivant comme un int et le renvoie. Apres avoir ete
lus, l'option et le parametre sont retires de la liste d'arguments.

/*-----------------------------------------------------------------*/
 int LitOptionDouble(Option, Double)
 char Option[];
 double *Double;

  renvoie 1 si l'option de nom Option est presente 0 sinon. Si oui,
lit l'argument suivant comme un double et le renvoie. Apres avoir
ete lus, l'option et le parametre sont retires de la liste
d'arguments.
```

```
/*-------------------------------------------------------------*/
```

B.1.3 Lecture des noms de fichier : LitOptionNomFichier()

```
/*-------------------------------------------------------------*/
 int LitOptionNomFichier(OptionFichier)
 char *NomFichier;

  renvoie 1 si un nom de fichier est present en option et 0 sinon. Si oui,
 le nom de fichier est renvoye. Par convention, les noms de fichiers :

 * ne sont precedes d'aucune Option.

 * ne commencent pas par le signe ''-''.

 * sont les arguments qui restent sur la ligne d'arguments apres que
toutes les autres options aient ete lues.

 Il faut donc appeler LitOptionNomFichier() APRES tous les autres
LitOption...().
/*-------------------------------------------------------------*/
```

B.1.4 Test si il reste une option non définie : YaPlusAucuneOption()

```
/*-------------------------------------------------------------*/
int YaPlusAucuneOption()

  renvoie 1 si il ne reste plus d'options, 0 sinon. Permet de tester
les erreurs de syntaxe sur la ligne d'argument. Par convention une
option est de la forme ''-option''.
/*-------------------------------------------------------------*/
```

B.2 Lectures et écritures au clavier sur le terminal

On propose ici quelques routines de lecture au clavier. Elles remplacent les scanf() dont elles évitent quelques lourdeurs, et quelques problèmes d'édition de ligne. Elles augmentent, en outre, la lisibilité du programme.
En fait, ces routines attaquent le input au niveau du ioctl(), et pas du scanf(), afin de pouvoir gérer le clavier au mieux.

B.2.1 Lecture d'un nombre : LitInt(), LitHex(), LitNbr(), LitDouble()

```
/*-------------------------------------------------------------*/
        int LitInt(Question)
        int LitHex(Question)
        int LitNbr(Question, min, max)
        double LitDouble(Question)
        char *Question;

 lit un entier au clavier apres avoir ecrit la Question sur l'ecran.
la fonction LitNbr() impose que l'entier soit compris entre min et max,
inclus,  lors de la lecture. LitInt() et  LitNbr() lisent un nombre
en base 10, LitHex lit un nombre en base 16.
```

```
/*-------------------------------------------------------------*/
```

B.2.2 Fabrication de menus : LitCh()

```
/*-------------------------------------------------------------*/
        char LitCh(Question)
        char *Question;
 lit un caractere au clavier apres avoir ecrit la Question sur l'ecran.

 Cette fonction est tres utilisee dans des menus dont voici un
modele :

 while(1) {
   switch(LitCh(
    "1 : Item1\n2 : Item2\n ...\nq : quit\nVotre Choix [1 2 ... q]")) {

    case 1 :
      .../...
    break;

    case 2 :
      .../...
    break;

      .../...

    case q:
      exit(0);
   }
 }

/*-------------------------------------------------------------*/
```

B.2.3 Lecture d'un choix binaire : OuiNon()

```
/*-------------------------------------------------------------*/
        int OuiNon(Question)
        char *Question;
 lit un O ou un N apres avoir ecrit la Question sur l'ecran. Renvoie 1
pour (O)ui et 0 pour (N)on.
/*-------------------------------------------------------------*/
```

B.2.4 Lecture d'une chaine de caractères : LitSt()

```
/*-------------------------------------------------------------*/
        LitSt(Question,Reponse)
        char *Question, Reponse[];
lit le string Reponse apres avoir ecrit la Question sur l'ecran.
/*-------------------------------------------------------------*/
```

B.2.5 Pause au sein d'un programme : pause() more()

```
/*-------------------------------------------------------------*/
```

```
        pause(message)
        char *message;
 ecrit le message et attend que l'utilisateur appuie sur une touche du
clavier pour rendre la main.

/*-----------------------------------------------------------------*/
        more()
 ecrit le message " --- more ---" et attend que l'utilisateur appuie
sur une touche du clavier pour rendre la main.
/*-----------------------------------------------------------------*/
```

B.2.6 Accès non standard au clavier : read_kbd()

```
/*-----------------------------------------------------------------*/
char read_kbd()
 attend et renvoie le caractere en entree.

Contrairement a scanf, stdin est gere caractere par caractere sans echo,
 c'est a dire que les caracteres arrivent :
> 1 a 1, et pas seulement apres un return
> ne sont pas affiches a l 'ecran
> ne sont pas filtres, sauf CTRL + C

  On simplifie donc les E / S interactives.
  Ce read_kbd() est bloquant (ie le programme s'arrete tant qu'un
caractere n'a pas ete frappe).
/*-----------------------------------------------------------------*/
```

B.2.7 Test si un caractère est entré au clavier : kbhit()

```
/*-----------------------------------------------------------------*/
int kbhit()
 renvoie 1 si il y a un caractere en entree et 0 sinon.

  Permet de detecter si l 'utilisateur a touche au clavier.
  Cet appel est non bloquant (ie la fonction retourne un argument
   meme si un caractere n'a pas ete frappe).
/*-----------------------------------------------------------------*/
```

B.2.8 Sortie de messages d'erreurs : erreur() warning()

```
/*-----------------------------------------------------------------*/
erreur(message)
warning(message)
char *message;

 Ecrit le message d'erreur passe en parametre, et dans le cas d'une
erreur au niveau d'Unix, recupere le message d'erreur et l'affiche.
erreur() fait un exit(-1), warning() rend la main.

/*-----------------------------------------------------------------*/
```

B.3 Lecture, écritures et traitements d'images et de données de vision

Les images sont généralement stockées au format *inrimage*. Voici des routines génériques qui permettent de les lire et de les écrire. Vu les tailles mémoire actuelles, on les écrit en bloc. Dans certaines implémentations, un autre format est utilisé.

B.3.1 Lecture d'une image codée en entier ou flottant : load_im() load_im_i() load_im_f()

```
/*--------------------------------------------------------------*/
load_im  (NomImage, Imc, TailleLigne, NombreLigne)
load_im_i(NomImage, Imi, TailleLigne, NombreLigne)
load_im_f(NomImage, Imf, TailleLigne, NombreLigne)
char NomImage[];
[unsigned] char Imc[TailleLigne][NombreLigne];
short int       Imi[TailleLigne][NombreLigne];
float           Imf[TailleLigne][NombreLigne];
int *TailleLigne, *NombreLigne;

 Lit une image inrimage codee en entier sur un octet (load_im()) ou
flottant sur 4 octets (load_im_f()) dans le fichier de nom NomImage.
En entree TailleLigne*NombreLigne et la taille du buffer Imc ou Imf
doit etre au moins egal a la taille de l'image lue. En sortie Imc ou
Imf contient les pixels, et TailleLigne, NombreLigne la taille reelle
de l'image.

 load_im_i() lit une image de caracteres et convertit le resultat
en short int.
/*--------------------------------------------------------------*/
```

B.3.2 Ecriture d'une image codée en entier ou flottant : store_im() store_im_f()

```
/*--------------------------------------------------------------*/
store_im(NomImage, Im, TailleLigne, NombreLigne)
store_im_f(NomImage, Imf, TailleLigne, NombreLigne)
char NomImage[];
[unsigned] char Im[TailleLigne][NombreLigne];
float           Imf[TailleLigne][NombreLigne];
int TailleLigne, NombreLigne;

 Ecrit une image inrimage codee en entier sur un octet (store_im()) ou
flottant sur 4 octets (store_im_f()) dans le fichier de nom NomImage.
L'image a NombreLigne lignes de TailleLigne pixels.
/*--------------------------------------------------------------*/
```

B.3.3 Extraction d'une sous-image : subima()

```
/*--------------------------------------------------------------*/
subima(ImSource, dimx, dimy, ImCible, nechx, nechy, x0, y0, pasx, pasy)
[unsigned] char ImSource[], ImCible[];
int dimx, dimy, x0, y0, nechx, nechy, pasx, pasy;

 Soit une image de taille dimx,dimy, on extrait de cette image un
```

```
 seuil l'image ImSource, et met le resultat dans l'image ImCible, les
pixels vaudront 0 si ils sont en dessous du seuil et 255 sinon.
Seuil est une valeur entre 0 et 255.

op1ima([signed] ImSource, [unsigned] ImCible, dimx,dimy, AbsIma)

 transforme une image signee en image non signee. Identique a
absima().

op1ima([unsigned] ImSource, [unsigned] ImCible, dimx,dimy, SmtIma(Fenetre))

 lisse l'image ImSource, et met le resultat dans l'image ImCible. Le
filtre employe est un filtre de Deriche modifie, la force du filtre
est specifie par une fenetre en nombre de pixel (Fen), le coefficient
du filtre etant ajuste pour que 90 cette fenetre.

op1ima([unsigned] ImSource, [unsigned] ImCible, dimx,dimy, EdgIma(Seuil))

 detecte les contours dans l'image ImSource, et met le resultat dans
l'image ImCible, les pixels vaudront 255 si c'est un point de contour
0 sinon. Le parametre de Seuil fixe une limite inferieure sur le
contraste. La methode employee est un detecteur de maximums de
contraste dans la direction du gradient. L'image source doit
etre lissee si il s'agit de donnees bruitees

 L'utilisation conjointe des deux routines precedentes constitue
un detecteur de contour Canny-Deriche.

op1ima([unsigned] ImSource, [unsigned->signed] ImCible, dimx,dimy, SousIma)
op2ima([unsigned] ImSource1, ImSource2, [signed] ImCible, dimx,dimy, SousIma)

 calcule la difference de deux images soit :
  ImSource - ImCible    -> ImCible
  ImSource1 - ImSource2 -> ImCible

 on notera que le resultat est divise par 2 afin que de valeurs entre
0 et 255 on obtienne des valeurs entre -128 et 127.

/*-------------------------------------------------------------*/
```

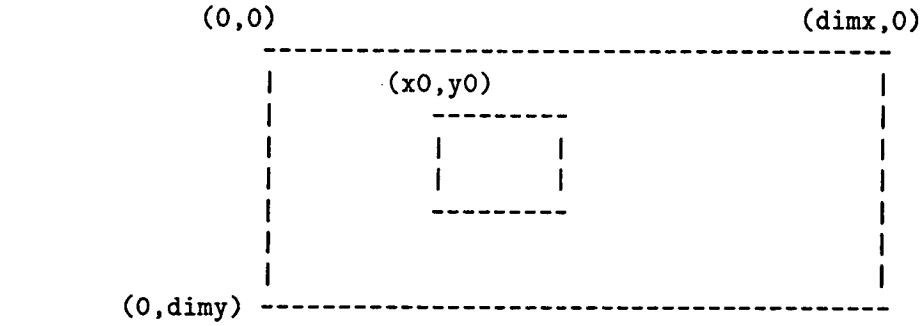
B.4 Routines mathématiques

On propose ici quelques routines de calcul habituellement utilisées dans les programmes. Certaines d'entre elles sont simplement des interfaces vers la bibliothèque mathématique *lnag*.
Elles ont été bien débuggées et ont toutes des routines de test, même les plus élémentaires. En les utilisant, on s'assure d'une certaine homogénéité dans les programmes.

B.4.1 Génération de nombres aléatoires : faleat() gauss()

```
/*-------------------------------------------------------------*/
double faleat()
```

```
rectangle a partir de la position (x0,y0), avec un pas
d'echantillonage de pasx sur les lignes et pasy sur les colonnes,
l'image finale sera de taille (nechx,nechy).
```




```
 Par exemple, avec :
  x0=0, y0=0, nechx=dimx, nechy=dimy/2, pasx=1, pasy=2
on extrait une trame non-entrelacee d'une trame entrelacee.
/*-------------------------------------------------------------------*/
```

B.3.4 Conversion d'images signees : absima()

```
/*-------------------------------------------------------------------*/
absima(ImSource, ImCible, dimx, dimy)
char ImSource[];
unsigned char ImCible[];
int dimx, dimy;

 Transforme une image signee en image non signee. La transformation
met en oeuvre une look-up table, et effectue une simple valeur
absolue.
/*-------------------------------------------------------------------*/
```

B.3.5 Operation sur des images : op1ima(), op2ima()

```
/*-------------------------------------------------------------------*/
op1ima(ImSource, ImCible, dimx, dimy, Operation)
op2ima(ImSource, ImSource2, ImCible, dimx, dimy, Operation)
[unsigned] char ImSource[], ImSource2[];
[unsigned] char ImCible[];
int dimx, dimy, Operation;

 Opere sur l'image(s) source(s) et met le resultat dans l'image cible.
Les images sont des tableaux de mots de 8 ebs interpretes comme des
nombres signes ou non signes selon l'operation. La particularite de
ses routines est de profiter d'une implementation plutot rapide en
travaillant par tables de valeurs et sur des images entieres, mais a
partir d'algorithmes de ''haut niveau''. Elles sont tres facilement
transportables.

 Les operations actuellement disponibles sont~:

op1ima([unsigned] ImSource, [unsigned] ImCible, dimx,dimy, ThrIma(Seuil))
```

```
 genere un nombre aleatoire entre 0.0 et 1.0
/*-----------------------------------------------------------------*/
double gauss()

 genere un nombre ale'atoire gaussien de moyenne 0.0 et d'ecart-type
1.0.
/*-----------------------------------------------------------------*/
```

B.4.2 Opérations entre matrices : addmat() addmulmat() clrmat() cpymat() crossprod() dotprod() jmulmat() linres() matdiag() mixtprod() mulmat() norm() normed() sousmat() scalmulmat() trp() tilde() tilded() rot_r2R() rot_R2r() writemat()

Voici les différentes routines de calcul sur les matrices et donc les vecteurs, formes linéaires et quadratiques que l'on utilise. Ces routines sont données avec un minimum de commentaires. Elles gèrent le cas où entrées et sorties se font dans la même mémoire.

```
/*-----------------------------------------------------------------*/
/* Addition de deux matrices (vecteurs)  :
   m3[dim1 * dim2] = m1[dim1 * dim2] + m2[dim1 * dim2]
*/

addmat(m1, m2, m3, dim1, dim2)
int dim1, dim2;

/*-----------------------------------------------------------------*/
/* Combinaison lineaire de deux matrices (vecteurs)  :

   m3[dim1 * dim2] = l1 m1[dim1 * dim2] + l2 m2[dim1 * dim2]
*/

addmulmat(l1, m1, l2, m2, m3, dim1, dim2)
int dim1, dim2;
double l1, l2, *m1, *m2, *m3;

/*-----------------------------------------------------------------*/
/* Initialisation d'une matrice a zero :

   m1[dim1 * dim2] = 0
*/

clrmat(m1, dim1, dim2)
double *m1;
int dim1, dim2;

/*-----------------------------------------------------------------*/
/* Copie d'une matrice vers une autre matrice :

   m2[dim1 * dim2] = m1[dim1 * dim2]
*/

cpymat(m1, m2, dim1, dim2)
double *m1, *m2;
```

```
int dim1, dim2;

double *m1, *m2, *m3;

/*-----------------------------------------------------------------*/
/* Produit vectoriel 3X3 :

   c[3] = a[3] ^ b[3]
*/

crossprod(a, b, c)
double *a, *b, *c;

/*-----------------------------------------------------------------*/
/* Produit scalaire de deux vecteurs :

 v = (a[dim] , b[dim])
*/

double dotprod(a, b, dim)
double *a, *b;
int dim;

/*-----------------------------------------------------------------*/
/* Transformation d'une matrice symmetrique par un jacobien :
                                                                  T
   m2[dim2, dim2] = J[dim2, dim1] * m1[dim1, dim1] * J[dim1, dim2]

> m1[] et m2[] sont des matrices symmetriques pleines non compactee
> J[] est la matrice d'un jacobien

*/
jmulmat(J, m1, m2, dim1, dim2)
double *J, *m1, *m2;
int dim1, dim2;

/*-----------------------------------------------------------------*/
int linres(A, x, b, Dimension)
double *A, *x, *b;
int Dimension;

 Effectue la resultion de l'equation     A * x = b

 ou b est un vecteur              de dimension Dimension
 ou x est un vecteur              de dimension Dimension
 ou A une matrice contigue, carree de dimension Dimension
 input :
               double A[Dimension][Dimension];
               double b[Dimension];
 output :
               double x[Dimension]; les sols si elles existent
```

```
 Renvoie 1 si la solution existe 0 si la matrice singuliere

/*-------------------------------------------------------------*/
int matdiag(M, D, U, Dimension)
double M[Dimension][Dimension], U[Dimension][Dimension], D[Dimension];
int Dimension;

 Effectue la diagonalisation de la matrice M telle que :
                    t
  M = U  * D * U
 ou D est un vecteur              contenant les valeurs  propres
 ou U est une matrice orthogonale contenant les vecteurs propres
 ou M DOIT ETRE une matrice symetrique, contigue, carree
  de dimension Dimension
 input :
                double M[Dimension][Dimension];
 output :
                double D[Dimension];
                double U[Dimension][Dimension];

 Les vecteurs sont ranges en colonne. C'est a dire que le n ieme
vecteur propre u[] sera donne par :

    u[0]         = U[n][0];
    u[1]         = U[n][1];
     ....
    u[Dimension] = U[n][Dimension];

 Renvoie 1 si la diagonalisation a ete faite et 0 si l'algorithme
ne converge pas.

/*-------------------------------------------------------------*/
/* Produit mixte de 3 vecteurs :

 v = | a[3], b[3], c[3] |
*/

double mixtprod(a, b, c)
double *a, *b, *c;

/*-------------------------------------------------------------*/
/* Multiplication de deux matrices :

   m3[dim1, dim3] = m1[dim1, dim2] * m2[dim2, dim3]
*/

mulmat(m1, m2, m3, dim1, dim2, dim3)
double *m1, *m2, *m3;
int dim1, dim2, dim3;

/*-------------------------------------------------------------*/
/* Norme d'un vecteur :
```

```
 v  = || a[dim] ||
*/

double norm(a, dim)
double *a;
int dim;

/*-----------------------------------------------------------------*/
/* Normalisation d'un vecteur :

 b[dim]  = a[dim] / || a[dim] ||
*/

normed(a, b, dim)
double *a, *b;
int dim;

/*-----------------------------------------------------------------*/
/* Calcul de la matrice de rotation R pour une rotation 3D d'axe
u[] et d'angle egal a la norme de u[]
              ^u[3]
   R[3,3] = e
*/

rot_r2R(r, R)
double r[3], R[3][3];

/*-----------------------------------------------------------------*/
/* Calcul inverse du precedent.
*/

rot_R2r(R, r)
double R[3][3], r[3];

/*-----------------------------------------------------------------*/
/* Multiplicationb d'une matrice par un scalaire :

   m2[dim1 * dim2] = alpha * m1[dim1 * dim2]
*/

scalmulmat(alpha, m1, m2, dim1, dim2)
double alpha, *m1, *m2;
int dim1, dim2;

/*-----------------------------------------------------------------*/
/* Soustraction de deux matrices (vecteurs)  :

   m3[dim1 * dim2] = m1[dim1 * dim2] - m2[dim1 * dim2]
*/

sousmat(m1, m2, m3, dim1, dim2)
```

```
int dim1, dim2;
double *m1, *m2, *m3;

/*-----------------------------------------------------------------*/
/* Transposee d'une matrice  :
                                    T
   m2[dim2, dim1] = m1[dim1, dim2]
*/

trp(m1, m2, dim1, dim2)
double *m1, *m2;
int dim1, dim2;

/*-----------------------------------------------------------------*/
/* Constitution d'une matride anti-symetrique a partir d'un vecteur 3D avec :
   H[3][3] x[3] = V[3] ^ x[3]
*/

tilde(v,H)
double v[3], H[3][3];

/*-----------------------------------------------------------------*/
/* Calcul inverse du precedent.
*/

tilded(H,v)
double H[3][3], v[3];

/*-----------------------------------------------------------------*/
/* Ecriture d'une matrice ou un vecteur avec son nom, sous une forme
facile a lire */

writemat(name, m, dimx, dimy)
char name[];
double m[];
int dimx, dimy;
/*-----------------------------------------------------------------*/
```

B.4.3 Opérations entre moyennes et covariances : cal_s() inv_s() inv_pos_sym_mat() jmulmats() mulvects() traces(), crossprod_12() distmaha() div_12() dotprod_12() fusion_12() normcrossprod_12() norm_12() normed_12() proj_12() rot_12() scalmulvect_12() writevec_12() DrawXEllipse12()

Les vecteurs sont representes par leurs valeurs moyennes de dimension n et leurs covariances qui est une matrice symmétrique stockée dans une vecteur de taille $n(n+1)/2$. On décrit les routines de calcul de ces quantités dans les expressions vectorielles.

```
/*-----------------------------------------------------------------*/
/* Transformation d'une matrice symmetrique en vecteur contenant
    ces n * (n+1) /2 composantes :

    ms[n(n+1)/2] = P(m[n,n])
        ( ms[0],                ... )
    M = ( ms[1], ms[2],         ... )
```

```
        ( ms[3], ms[4], ms[5], ... )
        ( ...                      )
*/
cal_s(m, ms, n)
double *m, *ms;
int n;

/*-----------------------------------------------------------------*/
/* Transformation inverse de cal_s() :
                                -1
    m[n,n] =  P(ms[n(n+1)/2])
*/
inv_s(ms, m, n)
double *ms, *m;
int n;

/*-----------------------------------------------------------------*/
int inv_pos_sym_mat(mat, inv, dimension)
double ap[dimension * (dimension+1) / 2];
double inv[dimension * (dimension+1) / 2];
int dimension;

 calcule l'inverse de la matrice mat[] de dimension specifiee dans la
matrice inv[]. Renvoie 1 si la matrice est definie positive, 0 sinon.
Dans ce dernier cas inv[] n'est pas defini. Les matrices symmetriques
doivent etre mises sous forme compactee (voir cal_s()).

 Il s'agit d'une routine d'inversion d'une matrice symmetrique definie
positive, typiquement une matrice de covariance.

/*-----------------------------------------------------------------*/
/* Transformation d'une matrice symmetrique par un jacobien :
 m2s = J  m1s  J^T
*/
jmulmats(J, m1s, m2s, dim1, dim2)
double *J, *m1s, *m2s;
int dim1, dim2;

/*-----------------------------------------------------------------*/
/* Multiplication par une matrice symetrique sous forme compressee
 y = Ms x
*/
mulvects(Ms, x, y, n)
double *Ms, *x, *y;
int n;

/*-----------------------------------------------------------------*/
/* Trace d'une matrice symetrique sous forme compressee
     _n
 v = >    Ms[i,i]
     -i=1
*/
```

```
traces(Ms, n)
double *Ms;
int n;

/*-----------------------------------------------------------------*/
/* Calcul de la moyenne et covariance d'un produit vectoriel
  z = x ^ y
  Cz = ~x Cy ~x^T + ~y Cx ~y^T
*/
crossprod_12(x, Cx, y, Cy, z, Cz)
double x[3], Cx[6], y[3], Cy[6], z[3], Cz[6];

/*-----------------------------------------------------------------*/
/* Distance de Mahanalobis entre deux vecteurs aleatoires
 d= (x-y)^T ( Cx + Cy )^(-1) (x-y)
*/
double distmaha(x, Cx, y, Cy, n)
double *x, *Cx, *y, *Cy;
int n;

/*-----------------------------------------------------------------*/
/* Calcul de la moy et var d'une division

 z= y/x;
 Vz= Vy/x^2 + Vx * y^2/x^4
*/
div_12(y, Vy, x, Vx, z, Vz)
double y, Vy, x, Vx, *z, *Vz;

/*-----------------------------------------------------------------*/
/* Calcul de la moyenne et covariance d'un produit scalaire
  z = x . y
  Cz = x^T Cy x + y^T Cx y
*/
dotprod_12(x, Cx, y, Cy, z, Cz, n)
double *x, *Cx, *y, *Cy, z[1], Cz[1];
int n;

/*-----------------------------------------------------------------*/
/* Fusion de deux mesures
 Cz = (Cx^-1 + Cy^-1)^-1
 z = Cz * ( Cx^-1 x + Cy^-1 y)
*/
fusion_12(x, Cx, y, Cy, z, Cz, n)
double *x, *Cx, *y, *Cy, *z, *Cz;
int n;

/*-----------------------------------------------------------------*/
/* Calcul de la moyenne et covariance de la norme d'un vecteur
  y = x / ||x||
  Vy = x^T Cx x
*/
```

```
norm_12(x, Cx, y, Vy, n)
double *x, *Cx, *y, *Vy;
int n;

/*-----------------------------------------------------------------*/
/* Calcul de la moyenne et covariance d'un vecteur norme
  y = x / ||x||
  Cy = 1 / ||x||^2 [ Cx - y y^T Cx y y^T ]
*/
normed_12(x, Cx, y, Cy, n)
double *x, *Cx, *y, *Cy;
int n;

/*-----------------------------------------------------------------*/
/* Calcul de la moyenne et covariance de la norme d'un produit vectoriel
  z = x ^ y
  Cz = ~x Cy ~x^T + ~y Cx ~y^T
*/
normcrossprod_12(x, Cx, y, Cy, z, Cz)
double x[3], Cx[6], y[3], Cy[6], z[1], Cz[1];

/*-----------------------------------------------------------------*/
/* Calcul de la moyenne et covariance de la projection perspective
   x[i]= x[i]/z;
*/
proj_12(M, CM, m, Cm)
double M[3], CM[6], m[2], Cm[3];

/*-----------------------------------------------------------------*/
/* Calcul de la moyenne et covariance d'un vecteur apres rotation (r petit)
  y = R x
  Cy = R Cx R^T + ~x Cr ~x^T
*/
rot_12(r, Cr, x, Cx, y, Cy)
double r[3], Cr[6], x[3], Cx[6], y[3], Cy[6];

/*-----------------------------------------------------------------*/
/* Calcul de la moyenne et covariance du produit par un scalaire
 y = l  x
 Cy = l^2  Cx + Vl x x^T
*/
scalmulvect_12(l, Vl, x, Cx, y, Cy, n)
double l, Vl, *x, *Cx, *y, *Cy;
int n;

/*-----------------------------------------------------------------*/
/* Ecriture de la moyenne et covariance d'un vecteur */
writevec_12(st, x, Cx, n)
char *st;
double *x, *Cx;
int n;
```

```
/*-------------------------------------------------------------*/
/* Trace d'une ellpsoide de covariance defini par :

          T    ( Cxx Cxy )        T
 m=(x0, y0)   C=(         )   (M-m)  C  (M-m) < 1
              ( Cxy Cyy )
*/
DrawXEllipse12(Win, x0, y0, Cxx, Cxy, Cyy, color)
ActiveXWindow *Win;
int x0, y0;
double Cxx, Cxy, Cyy;
int color;
/*-------------------------------------------------------------*/
```

B.4.4 Routines de résolution d'une itération du filtre de Kalman : kalvect()

L'équation de mesure du filtre de Kalman linéaire est utilisée pour la mise a jour de l'etat et de sa covariance grâce à cette routine. En voici la description :

```
/*-------------------------------------------------------------*/
kalvect(n, m, etat, mes, mat, var, covar)
int n, m;
double mes[m], mat[m][n], etat[n], var[m*(m+1)/2], covar[n*(n+1)/2];

  Entree :
 n     = dimension du vecteur d'etat etat
 m     = dimension du vecteur de mesure mes
 mat   : matrice de la relation mes = mat etat
 var   = covariance sur la mesure
 covar = covariance sur l'etat

  Attention : les deux matrices de covariance sont sous forme reduites
triangulaires, voir cal_s().

  Sortie :
 nouvel etat, nouveau covar

/*-------------------------------------------------------------*/
```

B.4.5 Tests statistiques et leurs distributions associées : t_student() F_fisher() pxi2()

On propose deux fonctions qui correspondent aux tables statistiques habituellement utilisées pour effectuer des tests sur de petits nombre d'échantillons. Le test de Student permet de déterminer si une distribution est significativement différente de 0 (avec toutes les variantes possibles), le test de Fisher permet de comparer deux variances.
Utilisées dans tests temps-réels, ces fonctions sont tabulées et interpolées, elles sont donc très rapides.

```
/*-------------------------------------------------------------*/
double t_student(n,prob)
int n;
double prob;
```

```
qui renvoie le seuil statistique de Student pour une distribution
a ''n'' degres de liberte et une probabilite d'erreur de ''prob''

les valeurs sont tabulees pour :
 prob = 0.9, 0.5, 0.3, 0.2, 0.1, 0.05, 0.02, 0.01, 0.001

/*-------------------------------------------------------------------*/
double F_fisher(p,q,prob)
int p,q;
double prob;

qui renvoie le seuil statistique de Fisher pour une distribution
a ''p,q'' degres de liberte et une probabilite d'erreur de ''prob''
avec :

F= sp^2/sq^2

les valeurs sont tabulees pour :
 prob = 0.05, 0.025, 0.01, 0.001

/*-------------------------------------------------------------------*/
```

Pour des tests sur les distributions gaussiennes, cette routine n'est pas interpolee :

```
/*-------------------------------------------------------------------*/
double pxi2(xi2,n)
double xi2;
int n;

 renvoie la probabilite pour que la valeur xi2 correspondant a une
distribution Xi^2 a n degres de liberte soit significativement
non-nulle.

/*-------------------------------------------------------------------*/
```

B.4.6 Filtrage médian monodimensionnel : fmedian()

Cette routine de filtrage médian monodimensionnel a été optimisée pour son utilisation en temps réel, elle effectue du tri rapide des échantillons et gère leur entrée-sortie par des tables d'index. Il y a bien sûr un déphasage égal à la moitié de la taille de la fenêtre de par le principe de ce filtre.
Elle s'utilise trivialement, sur une image.

```
/*-------------------------------------------------------------------*/
int fmedian(x, FiltreMedian)
int x;
int *FiltreMedian;

 renvoie la valeur filtree de l'echantillon x, pour le filtre
FiltreMedian. Avant le premier appel le pointeur FiltreMedian doit
etre initialise avec la routine suivante, il ne doit pas etre modifie
ensuite.

 La routine fmedian() doit etre appellee sequentiellement sur tous les
echantillons du signal. La structure de donnee reliee a FiltreMedian
contient les valeurs des echantillons precedents, utilises pour le
```

```
filtrage.

/*-----------------------------------------------------------------*/
int *CreeFiltreMedian(DemiTailleFenetre)
int DemiTailleFenetre;

 Cette routine cree une structure de donnee permettant de gerer un filtre
median sur une fenetre de taille +/-DemiTailleFenetre.

/*-----------------------------------------------------------------*/
```

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique.