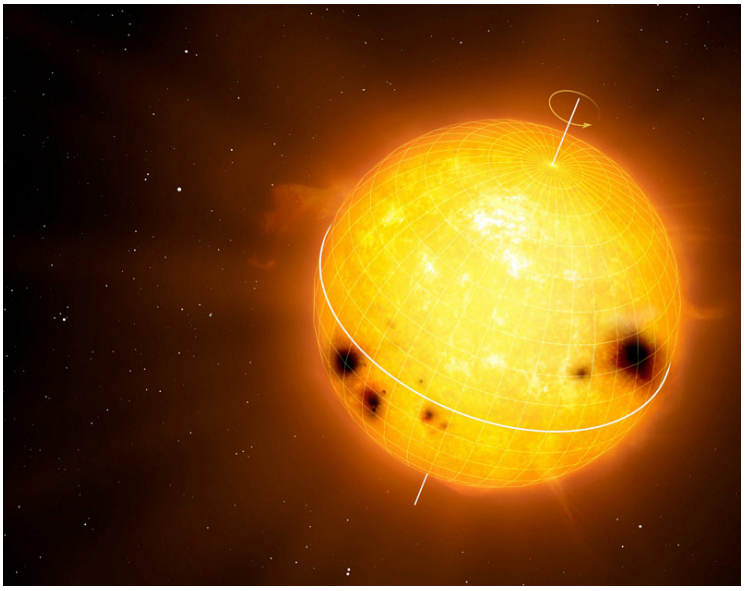
Master Astronomie et Astrophysique

Observatoire de Paris

Projet Informatique en langage C

Mesure de la rotation différentielle du Soleil



Florence Henry, LESIA, florence.henry@obspm.fr

Benjamin Godard, LERMA, benjamin.godard@obspm.fr

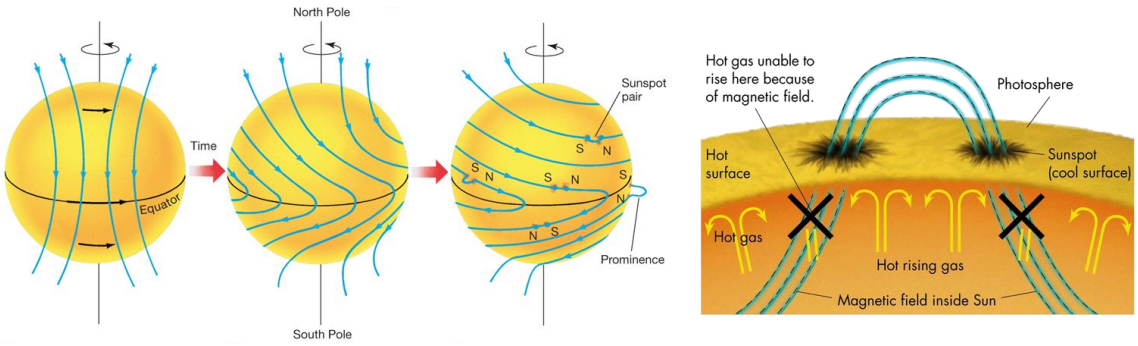
Année 2017 – 2018

Table des matières

Chapitre 1

Objectifs du projet

1.1 Contexte scientifique



L'essentiel de la lumière visible qui nous vient du Soleil est émis par la photosphère. C'est la couche la plus profonde de l'atmosphère et également la zone la plus froide, entre 6000 K et 4200 K. Elle s'étend sur environ 330 km de profondeur. La photosphère est composée de granules plus brillantes et plus chaudes que la moyenne de la photosphère entourées de zones inter-granules plus froides. Ces structures sont les témoins de la convection de la matière sous la surface visible. La durée de vie des granules est de l'ordre d'une dizaine de minutes.

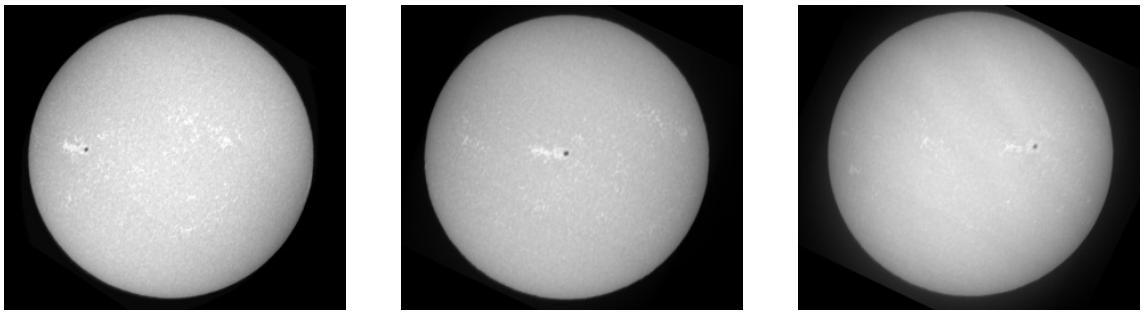
En plus de ces structures, on distingue des zones beaucoup plus sombres et donc froides (environ 3900 K), de quelques milliers à quelques dizaines de milliers de kilomètres, appelées taches solaires. Elles sont le résultat de la déformation de boucles du champ magnétique interne par la rotation différentielle du Soleil et atteignent l'atmosphère. Le champ magnétique intense dans les taches bloque le mouvement convectif et diminue les apports d'énergie, ce qui explique l'absence de granule et la température plus basse.

Les taches apparaissent généralement par couple de polarités opposées, les lignes de champ magnétique semblant se boucler de l'une à l'autre. Il existe également des taches isolées ainsi que des groupes complexes pouvant contenir plus d'une dizaine de taches.

La durée de vie des taches varie de un à quelques mois, ce qui permet de suivre leur évolution sur plusieurs rotations solaires. C'est cette durée de vie qui va nous permettre de mettre en évidence et de mesurer la rotation différentielle du Soleil.

1.2 Objectif scientifique

Le but du projet est de mettre en évidence et de mesurer la rotation différentielle du Soleil en suivant l'évolution de taches solaires sur un jeu d'images de la photosphère.



2005-09-20 15:57:54.000 2005-09-23 08:35:25.000 2005-09-25 07:16:39.000

Pour cela, vous utiliserez des images prises au spectrohéliographe de Meudon. Cet instrument utilise des filtres très étroits (dit filtres interférentiels) qui permettent d'observer le Soleil à une température particulière et donc, à une profondeur donnée. Le filtre utilisé pour ces clichés permet d'observer la zone de la photosphère où les taches sont bien visibles (bande CaII K1v proche de la raie du calcium une fois ionisé).

1.3 Plan de développement

Le principe est de partir d'un code de base qui vous est fourni, et de le complexifier au fur et à mesure pour obtenir le programme qui résoud le problème scientifique posé.

Les étapes principales du développement seront les suivantes :

1. recherche des images ;
2. affichage d'une image ;
3. réglage du contraste ;
4. ré-échantillonnage de l'image ;
5. recadrage de l'image ;
6. calcul de la latitude et de la longitude de la tache ;
7. transformation du programme en procédure ;
8. application à plusieurs images successives - suivi d'une tache ;
9. calcul de la vitesse de la tache, puis de sa période de rotation ;

10. calculs des vitesses des taches situées à différentes latitudes ;
11. conclusions.

1.4 Recherche des images

Les images du spectohéliographe de Meudon sont regroupées dans la base de données BASS2000 : http://bass2000.obspm.fr/home.php?lang=fr.

Pour récupérer des images acquises à une certaine date (celle de votre anniversaire par exemple), il suffit d'aller dans la section "Rechercher des observations" et de remplir le formulaire. N'oubliez pas que ce sont les images de la bande K1v qui nous intéressent.

> Lorsque vous avez trouvé une image où les taches ont un bon contraste, remontez dans le temps pour trouver la première image où la tache apparaît. Puis avancez dans le temps pour trouver la dernière image de la tache. Entre ces 2 dates, prenez un nombre suffisant d'images pour avoir une mesure précise de la rotation du Soleil.

Note sur le format des images

Dans la base de données BASS2000, les images sont disponibles dans 2 formats :

GIF : format où les images sont compressées mais dégradées ;

FITS : format très couramment utilisé en astrophysique et qui est composé :

- d'un en-tête (*header*) au format texte. Celui-ci comporte plusieurs mots-clés (*keyword*) et leur valeur associée, décrivant les paramètres de l'image : la date d'observation, la longueur d'onde, etc...
- l'image en binaire ensuite.

Les images au format GIF sont utiles pour faire votre sélection.

Les images au format FITS sont celles sur lesquelles vous aller travailler. Vous aurez en effet besoin des informations du *header*, ainsi que d'une haute définition pour les calculs.

Chapitre 2

Développement du code

2.1 Code de base

2.1.1 Description

Vous disposez au départ d'un code de base qui lit et affiche une image au format FITS, demande à l'utilisateur de cliquer sur un point de l'image, et renvoie les coordonnées en pixels de ce point. Ce code de base est composé de quatre fichiers :

```
affiche.c      affiche.h
affiche_lib.c  affiche_lib.h
```

Le fichier `affiche.c` (voir annexe A) est le code source principal. Pour réaliser les tâches ci-dessus, celui-ci fait appel à deux fonctions contenues dans `affiche_lib.c` et dont vous pouvez trouver le mode d'emploi en Annexe (page 25) :

- `LectureFITS`
- `AfficheFITSEtLitPixel`

Le fichier `affiche_lib.c` contient les fonctions précédentes ainsi que deux autres fonctions utilitaires. Vous n'aurez pas à toucher à ce fichier, mais juste à le compiler.

2.1.2 Librairies nécessaires

Afin de lire les données au format FITS mais aussi d'interagir avec l'environnement graphique de l'ordinateur, les routines de `affiche_lib.c` font appel à des fonctions issues de deux librairies[1] spéciales : `libX11.so` et `libcfitsio.so`.

1. rappel : une librairie est une collection de routines compilées

- Sous Unix, l'affichage à l'écran est géré par le *serveur X*[2]. La librairie `libX11.so` contient des fonctions qui permettent de dialoguer avec ce serveur, et donc d'afficher des fenêtres à l'écran, et des images à l'intérieur de celles-ci.
- Les fonctions de la librairie `libcfitsio.so` permettent, quant à elles, de lire et d'écrire des fichiers au format FITS, c'est-à-dire d'interpréter les données de l'en-tête ainsi que de lire l'image stockée en binaire.

2.1.3 Compilation du code

Pour qu'un programme puisse utiliser des librairies, il faut spécifier au moment de la compilation où se trouvent les librairies en question (voir cours, chap. 2.4). Pour cela, on utilise les options `-l`, `-L` et `-I` du compilateur `gcc`.

`-l<nom>` indique qu'il faut lier la librairie `lib<nom>.so` au programme.

`-L<chemin>` indique qu'il faut chercher les librairies dans le répertoire `<chemin>`.

`-I<chemin>` indique qu'il faut chercher les en-têtes dans le répertoire `<chemin>`.

Votre programme utilise les librairies `libm.so` et `libcfitsio.so`.

Ces librairies sont contenues dans le répertoire `/usr/lib`.

Leurs fichiers d'en-têtes (e.g. `fitsio.h`) sont dans le répertoire `/usr/include`.

Le programme étant composé de deux fichiers source, la compilation doit s'effectuer de la manière suivante (voir cours, chap. 6.2.4) :

```
gcc -ansi -Wall -Wextra -I/usr/include -c affiche.c
gcc -ansi -Wall -Wextra -I/usr/include -c affiche_lib.c
gcc -L/usr/lib -o affiche affiche.o affiche_lib.o -lm -lX11 -lcfitsio
```

Note : le compilateur connaît déjà par défaut les chemins `/usr/lib` et `/usr/include`. Il n'est donc pas nécessaire de les indiquer dans les lignes de commandes ci-dessus.

En développant le projet, vous allez écrire de nouveaux fichiers. A l'instar de `affiche.c` et `affiche_lib.c`, chaque fichier devra être compilé séparément, et les fichiers objets (`.o`) liés ensemble au moment de la production de l'exécutable.

Pour un programme contenant des centaines de fichiers ce travail peut vite devenir fastidieux. En outre, le programmeur doit constamment garder en mémoire les fichiers qu'il a modifiés et qu'il faut recompiler. Pour résoudre ces problèmes et simplifier / structurer le travail de programmation, on utilise un **Makefile**.

2. Voir http://fr.wikipedia.org/wiki/X_Window_System pour plus de détails sur son fonctionnement.

2.2 Utilisation d'un Makefile

Le Makefile[3] est un fichier qui décrit un ensemble d'actions, comme la compilation d'un projet, l'archivage de document, etc. Il est lu avec la commande `make`[4] qui interprète les actions décrites et les exécute.

2.2.1 Les règles dans un Makefile

Un Makefile est composé de plusieurs règles qui s'écrivent comme ceci :

```
nom_cible : dependance_1 dependance_2 ...
    commande a executer
    commande a executer
    ...
```

Un Makefile s'utilise via la commande `make nom_cible`. La règle dont le nom est spécifié est évaluée. Si aucun nom n'est donné, la première règle du Makefile est évaluée.

- Les dépendances sont soit des noms de fichiers, soit des noms d'autres règles.
- Quand une règle est évaluée, les dépendances sont analysées. Si une dépendance est la cible d'une autre règle, cette régle est à son tour évaluée (et la première règle est mise en attente).
- Une fois les dépendances analysées, si l'une des dépendances est plus récente que la règle, les commandes qui suivent sont exécutées.
- Chaque commande doit être précédée d'une `<tabulation>`. Une commande peut être écrite sur plusieurs lignes à condition de terminer chaque ligne par \

Ainsi, la commande `make` n'effectue pas nécessairement les instruction d'une cible. Seule une cible dont les dépendances ont changé depuis la dernière utilisation est traitée.

Exemple

```
affiche : affiche.o affiche_lib.o
   gcc -o affiche affiche.o affiche_lib.o -lm -lX11 -lcfitsio

affiche.o : affiche.c affiche.h affiche_lib.h
   gcc -ansi -Wall -Wextra -c affiche.c

affiche_lib.o : affiche_lib.c affiche_lib.h
   gcc -ansi -Wall -Wextra -c affiche_lib.c
```

3. Pour plus de détails sur les Makefile, voir http://gl.developpez.com/tutoriel/outil/makefile/
4. la commande `make` lit le fichier appelé « Makefile » situé dans le répertoire courant.

2.2.2 Les variables dans un Makefile

Pour simplifier le Makefile, le rendre plus lisible et plus facile à maintenir, il est enfin possible de définir des variables. Celles-ci peuvent contenir des nombres ou des chaînes de caractères, comme par exemple les chemins d'accès aux librairies ou encore le nom du compilateur... Si l'on souhaite changer de compilateur, il suffit alors de changer uniquement la valeur de la variable correspondante.

Une variable se déclare sous la forme `NOM=VALEUR` et s'utilise via `$(NOM)`.

Exemple

```
# ============================================================================== #
#                                 Variables du Makefile                          #
# ============================================================================== #

# ----------------------------------------
# 1 - variable du compilateur a utiliser
# 2 - choix des options de compilation
# 3 - liste des repertoires d'en-tetes
# 4 - liste des repertoires de librairies
# 5 - librairies a lier
# ----------------------------------------
CC     = /usr/bin/gcc
CFLAGS = -ansi -Wall -Wextra -Werror
IFLAGS = -I /usr/include
LFLAGS = -L /usr/lib
LINK   = -lX11 -lcfitsio -lm

# ============================================================================== #
#                                   Cibles du Makefile                           #
# nom_cible : dependance                                                         #
# <tabulation> commande                                                          #
# ============================================================================== #

# ----------------------------------------
# ! - all : Compile tous les executables
# ! - help : Affiche cette aide
# ! - clean : efface les fichiers objet *.o
# !           et les executables
# ----------------------------------------
all : affiche

help :
   @grep -E "^# !" Makefile | sed -e 's/! -/-/g'

clean :
   rm -f *.o affiche

# ----------------------------------------
# Executables
# ----------------------------------------
affiche : affiche.o affiche_lib.o
   $(CC) $(LFLAGS) affiche.o affiche_lib.o -o affiche $(LINK)

# ----------------------------------------
# Fichiers intermediaires (*.o)
# ----------------------------------------
affiche.o : affiche.c affiche.h affiche_lib.h
   $(CC) $(CFLAGS) $(IFLAGS) -c affiche.c

affiche_lib.o : affiche_lib.c affiche_lib.h
   $(CC) $(CFLAGS) $(IFLAGS) -c affiche_lib.c
```

2.3 Prise en main

L'archive qui vous a été donnée contient un Makefile ainsi qu'une image du soleil au format FITS : `mk140505.065538.fits`.

- Compilez le programme en vous servant du Makefile. Testez-le sur l'image fournie et comprenez le fonctionnement des fonctions appelées par `affiche.c`.
- Familiarisez-vous avec le Makefile. Essayez les différentes cibles et comprenez leurs objectifs. Apportez des modifications et testez les conséquences.

Chapitre 3

Traitements à effectuer

3.1 Planning et consignes

Le projet se subdivise en six étapes successives, décrites plus bas. Afin de pouvoir finir dans les temps, il est conseillé de suivre le planning suivant

- séance 1 — régler le contraste de l'image
- ▷ *hors séance* — trouver un jeu d'images
ré-échantillonner l'image
- séance 2 — recadrer l'image
calculer la latitude et la longitude d'une tache
- séance 3 — transformer le programme en sous-programme
calculer la vitesse de plusieurs taches
- ▷ *hors séance* — écrire le rapport

A la fin du projet, vous soumettrez sur la plateforme deux fichiers séparés, **votrenom-rapport.pdf**, et l'archive **votrenom-source.tar** contenant uniquement les fichiers sources de votre programme (aucune image). L'archive devra contenir les fichiers suivants

`affiche.c`	`affiche.h`	`echantillon.c`	`echantillon.h`
`affiche_lib.c`	`affiche_lib.h`	`main.c`	`Makefile`
`contraste.c`	`contraste.h`	`recadrage.c`	`recadrage.h`
`coordonnees.c`	`coordonnees.h`	`reglin.c`	`reglin.h`

Chaque fichier ci-dessus est associé à une étape. Dans chaque fichier, vous privilégierez la construction de fonctions pour réaliser les différents traitements demandés.

3.2 Réglage du contraste de l'image

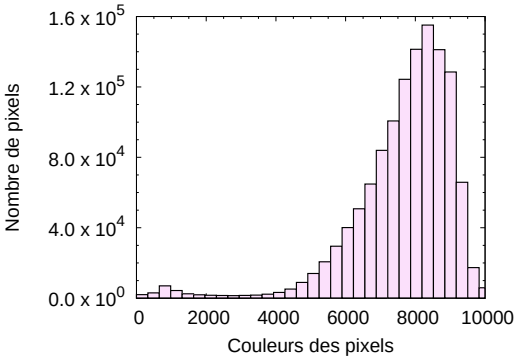
La fonction `AfficheFITSEtLitPixel` est appelée, dans le code de base, avec `min = 0`, et `max = 32767`, c'est-à-dire que le noir est associé à la valeur 0 et le blanc à 32767.

Comme l'intensité du soleil s'étend sur une moins grande gamme de valeurs, ses pixels les plus clairs apparaissent en gris foncé et ceux les plus sombres en gris très foncé.

La première étape consiste à régler l'échelle afin d'afficher tous les détails du Soleil, i.e. que le noir corresponde à ses pixels les plus sombres et le banc aux pixels les plus clairs.

Histogramme de couleur

Si on réalise l'histogramme des couleurs d'une image, i.e. si l'on compte le nombre de pixels dans des intervalles de couleur, on obtient un graphique tel que ci-contre.

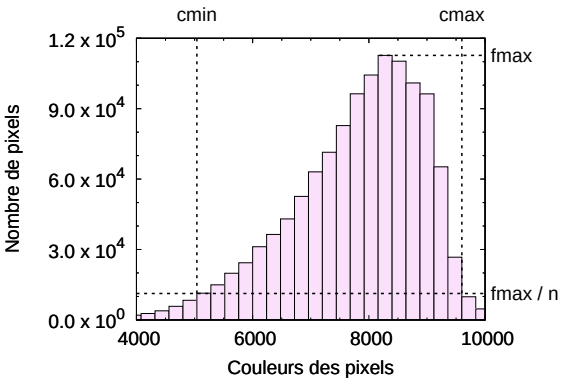


La première bosse (~ 1000) correspond aux pixels du fond du ciel. La deuxième correspond aux pixels du Soleil. On voit ici que les pixels qui nous intéressent ont des couleurs dont la valeur est entre 4000 et 10 000.

> Réalisez l'histogramme de l'image. Pour cela, créez un tableau de 100 entiers, où l'élément `i` contiend le nombre de pixels de l'image qui vérifient
> `i * 32767 / 100 <= image(j,k) < (i+1) * 32767 / 100`
>
> On peut obtenir un histogramme de meilleure résolution en prenant en compte le fait que le maximum de l'image n'est pas à 32767. Comment ?

Analyse de l'histogramme

Si l'on appelle `AfficheFITSEtLitPixel` avec `min = 4000`, et `max = 10000`, le contraste est meilleur. Mais ces chiffres ne sont pas les mêmes d'une image à l'autre. Il faut donc les trouver automatiquement.



Pour déterminer la gamme de couleurs intéressantes, il faut chercher la couleur la plus fréquente, i.e. le maximum de l'histogramme $f_{\max}$. Une fois $f_{\max}$ trouvé, on considère que

les couleurs qui nous intéressent sont celles qui ont une fréquence supérieure à $f_{\max}/n$ (voir graphique) où n est un nombre à déterminer.

Cette méthode peut toutefois soulever un problème : dans certaines images, un très grand nombre de pixels peuvent se trouver dans les zones sombres (en dehors du soleil), ce qui peut fausser la détermination de $f_{\max}$. Pour éviter cela, on peut mettre à zéro les éléments de l'histogramme correspondant aux couleurs inférieures à une valeur limite.

Mettez à 0 toutes les éléments de l'histogramme correspondant à des couleurs inférieures à une valeur limite de 1000. Calculez la couleur la plus fréquente.

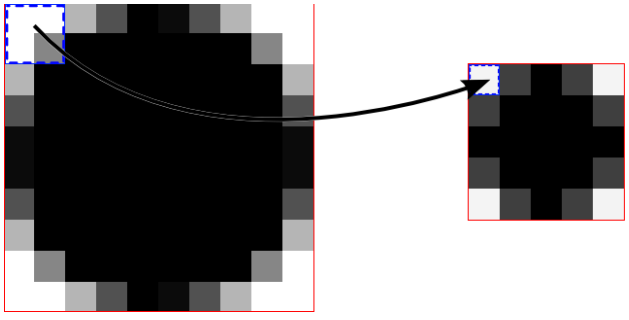
Fixez la valeur de n et calculez les couleurs minimale et maximale ($c_{\min}$ et $c_{\max}$ sur le graphique). Testez différentes valeurs de n et trouvez celle qui optimise le contraste.

Comment pourriez-vous faire pour déterminer n automatiquement ? (il n'est pas demandé de coder ceci, mais juste de donner des idées).

3.3 Ré-échantillonnage de l'image

La plupart des images sur BASS2000 font plus de 1500 pixels de haut, ce qui est plus grand que les écrans que vous utilisez. Il faut donc réduire la taille des images, afin de pouvoir visualiser le Soleil dans son intégralité.

Appliquer un facteur de réduction `ech` à une image est simple. Si l'image initiale mesure `N1 x N2`, l'image finale mesure `N1/ech x N2/ech`. Chaque pixel de l'image finale est la moyenne d'un carré de `ech x ech` pixels de côté. La figure ci-dessous montre un exemple pour `ech` = 2. Le pixel de coordonnées $(0, 0)$ de l'image finale est la moyenne des 4 pixels de coordonnées $(0, 0)$, $(0, 1)$, $(1, 0)$ et $(1, 1)$ de l'image initiale.



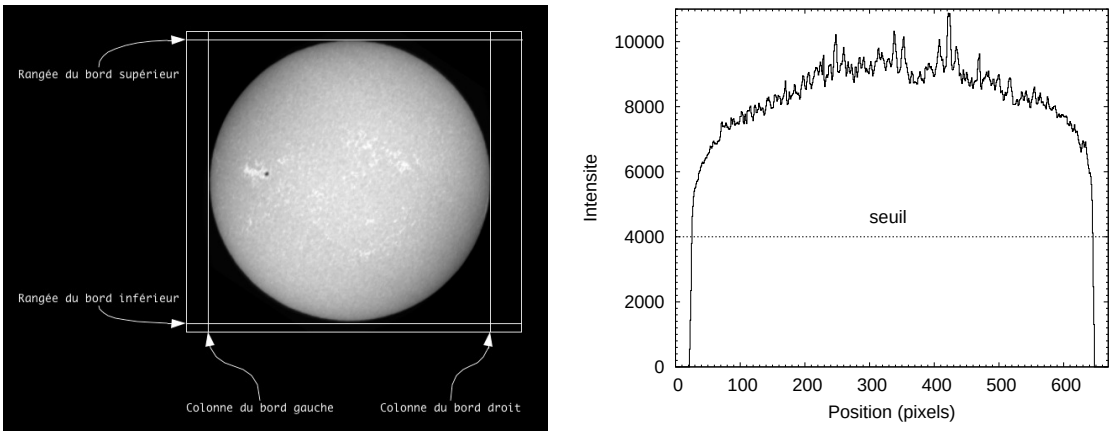
Définissez un facteur de réduction entier et fixez sa valeur à 2.
Créez un nouveau tableau qui contient la nouvelle image.

3.4 Recadrage de l'image

Pour pouvoir calculer la latitude et la longitude d'une tache d'après ses coordonnées (x, y), il faut connaître les coordonnées du centre du Soleil et son rayon.

Or, les images sélectionnées ne montrent pas toujours le Soleil avec le même nombre de pixels, et ce dernier n'est pas forcément toujours au même endroit dans l'image.

Il faut donc faire une recherche des bords inférieur, supérieur, gauche et droit du Soleil et recadrer l'image pour que les bords du Soleil correspondent aux bords de l'image, comme le montre la figure ci-dessous (panneau de gauche).



Notez que les bords d'un astre ne sont jamais nets : le passage entre le fond du ciel et l'astre est progressif, comme le montre le profil de la figure ci-dessus (panneau de droite).

Pour déterminer le bord d'un astre, il faut donc fixer un seuil à partir duquel on considère que le bord est franchi, comme par exemple, 30% du maximum de l'image.

> Déterminez les bords du Soleil. Modifiez le seuil et observez les changements.
> Rangez la partie intéressante de l'image dans un tableau de taille adaptée au Soleil.
> Affichez cette image à la place de l'image FITS du départ.

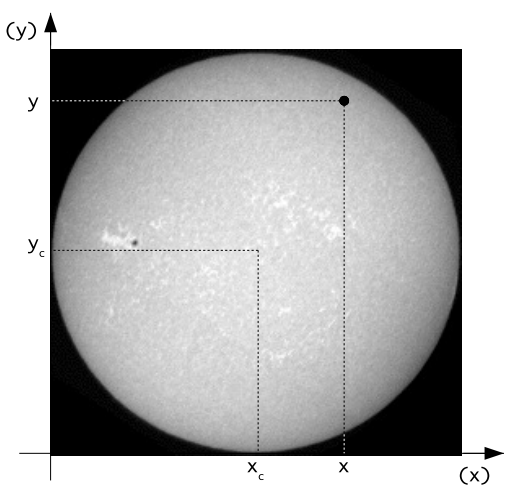
3.5 Calcul de la latitude et longitude de la tache

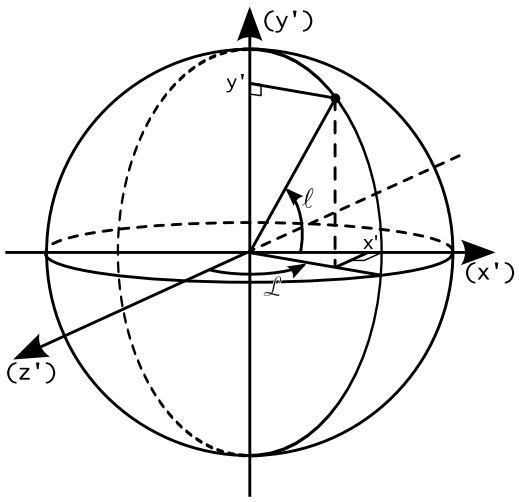
Cliquer sur une tache affiche ses coordonnées en pixels, l'origine du repère étant le coin supérieur gauche. Pour calculer la latitude et la longitude associées à ces coordonnées cartésiennes, il faut tout d'abord translater le centre du repère vers le centre du Soleil, de coordonnées (x_c, y_c).

Calculez les coordonnées (x', y') de la tache dans le repère centré sur (x_c, y_c).

Transformez les coordonnées (x', y') ainsi obtenues en coordonnées sphériques $(R, \mathscr{L}, l)$, où $\mathscr{L}$ est la longitude, l la latitude et R le rayon du Soleil en pixels.

Afin que tout le monde ait les mêmes notations, il est demandé de suivre **impérativement** les notations des axes de la figure ci-dessous :





3.6 Transformation en sous-programme

Transformez votre programme en une fonction qui prend en argument le nom du fichier à lire, et renvoie l'heure d'observation en secondes et les coordonnées (latitude et longitude $\mathscr{L}$ et l) de la tache.

Écrivez un nouveau programme principal qui demande à l'utilisateur le nombre d'images à lire, et leurs noms. Utilisez des tableaux pour stocker, pour chaque image, l'heure et les coordonnées calculées par la fonction.

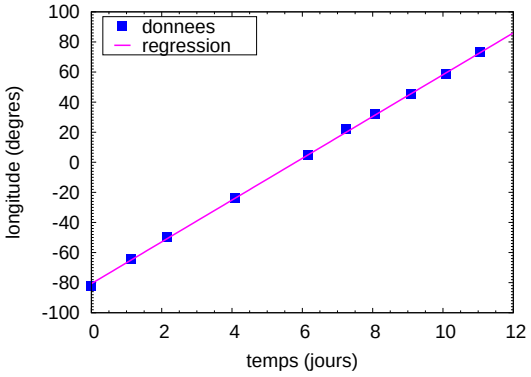
Écrivez le nombre d'images et les noms des images dans un fichier texte. Redirigez le contenu de ce fichier vers l'entrée standard, afin d'appeler le programme ainsi

```
./programme.x < donnees.dat
```

3.7 Calcul de la période de rotation

L'analyse des coordonnées successives d'une tache montre que la latitude ne change pas (ou peu) en fonction du temps. Pourquoi ?

La vitesse d'une tache est constante. Tracer sa longitude en fonction du temps conduit donc à des points alignés sur une droite (voir figure) dont la pente correspond à la vitesse de la tache.



Calculez la vitesse d'une tache en utilisant la régression linéaire programmée en TD. Réitérez ce calcul pour une tache située à une autre latitude (vous prendrez si besoin un autre jeu d'images). Qu'en concluez-vous ?

Comparez vos résultats aux périodes de rotation du Soleil trouvées sur internet.

3.8 Écriture du rapport

Le rapport fera entre 1 et 4 pages et **ne doit pas** paraphraser le sujet du TP.

1. **Introduction -** Expliquez en 1 page max ce que vous avez compris du but du TP.
2. **Déroulement -** Présentez les différentes étapes de la réalisation du TP (2 pages). Quelles sont les principales difficultés que vous avez rencontrées ?
3. **Présentation des résultats -** Donnez les résultats que vous avez obtenus (1 page). Sont-ils en accord avec la théorie ? Ayez un regard critique envers vos résultats.

Vous pourrez ensuite écrire quelques paragraphes sur les points suivants. Il ne s'agit pas de coder, mais de réfléchir et de donner les idées que vous avez.

1. **Calcul des incertitudes (1/3) -** Le programme de régression linéaire vous donne une erreur sur la pente de la droite trouvée. Utilisez cette valeur pour calculer l'erreur sur la période de rotation. Cette erreur vous semble-t-elle réaliste ?
   Rappel : lorsque 2 variables T et a sont liées par une fonction f :

   $$T = f(a)$$

   alors l'incertitude sur T, ΔT, se calcule ainsi en fonction de Δa :

   $$\Delta T = f'(a) \times \Delta a$$

2. **Calcul des incertitudes (2/3) -** De combien de pixels varie le rayon du Soleil lorsque vous changez le seuil ? Déterminez la précision sur le calcul du rayon.

3. **Calcul des incertitudes (3/3) -** Si vous lancez le programme plusieurs fois sur la même image, vous constatez que vous ne cliquez jamais exactement au même endroit. De combien de pixels déviez-vous ? Calculez l'incertitude sur le calcul de la longitude et de la latitude (n'oubliez pas l'erreur sur le rayon). En déduire l'incertitude sur la période de rotation.

Annexe A

Programme de base : affiche.c

```c
/* ========================================================================================== */
/* ====================== Description du programme affiche.c ========================== */
/* ========================================================================================== */
/* Auteurs & modifications                                                                    */
/*    - Florence HENRY  : Novembre 2011                                                       */
/*    - Benjamin GODARD : Fevrier  2016                                                       */
/*                                                                                            */
/* Fonctionnement                                                                             */
/*    - Lit un fichier .fits                                                                  */
/*    - Affiche l'image a l'ecran                                                             */
/*    - Attend : * un clic de souris pour renvoyer les coordonnees du pixel                   */
/*               * l'appui sur la lettre q pour fermer la fenetre                             */



/* ========================================================================================== */
/* ============================== En tetes necessaires ============================== */
/* ========================================================================================== */
#include "affiche.h"
#include "affiche_lib.h"
#include <stdio.h>
#include <string.h>



/* ========================================================================================== */
/* ============================== Definitions de fonctions ============================== */
/* ========================================================================================== */



/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/* Fonction principale du programme                                                           */
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
int main ( void )
{
   /* ------------------------------------------------------ */
   /* Variables necessaires a l'affichage de l'image         */
   /* ------------------------------------------------------ */
   char   filename[CAR] = {'\0'} ; /* nom de l'image - FITS */
   long **cube           = NULL  ; /* tab contenant l'image */
   long   dim1           = 0     ; /* dim de l'image Nx     */
   long   dim2           = 0     ; /* dim de l'image NY     */
   int    x              = 0     ; /* coordonnee x du pixel */
   int    y              = 0     ; /* coordonnee y du pixel */
   long   nb_sec         = 0     ; /* Heure d'obs, en sec   */
```

```
   /* ------------------------------------------------------ */
   /* Lecture de l'image                                     */
   /* ------------------------------------------------------ */
   printf ( "Entrez le nom de l'image a lire:\n" ) ;
   fgets ( filename, CAR-1, stdin ) ;
   supp_saut_ligne ( filename ) ;

   cube = LectureFITS ( filename, &dim1, &dim2, &nb_sec ) ;
   printf ( "L'image fait %ld (colonnes) par %ld (lignes) pixels\n", dim1, dim2 ) ;

   /* ------------------------------------------------------ */
   /* nbsec contient le nb de secondes depuis le 1/1/1970    */
   /* verifier la coherence avec la commande unix            */
   /* date -r <contenu de nbsec>                             */
   /* ------------------------------------------------------ */
   printf ( "Un clic sur l'image affiche les coordonnees du pixel et sa valeur.\n" ) ;
   printf ( "Appuyer sur la touche 'q' pour sortir.\n" ) ;
   AfficheFITSEtLitPixel ( cube, dim1, dim2, 0, 32767, &x, &y ) ;

   return 0 ;
}



/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/* La fonction suivante permet de vider le tampon associe a un flot                   */
/*    -> l'adresse du flot est donnee en argument par l'utilisateur                   */
/*    -> la fonction ne renvoie rien. Aucune erreur n'est prise en compte             */
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
void viderbuffer ( FILE *stream )
{
   char cc = 0 ;
   while (cc != '\n' && cc != EOF)
   {
      cc = fgetc(stream) ;
   }
   return ;
}



/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
/* La fonction suivante permet de remplacer '\n' par '\0' dans une chaine de caractere  */
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
void supp_saut_ligne ( char *chaine )
{
   char *c = NULL ;
   if( (c = strchr(chaine,'\n')) != NULL )
   {
      *c = '\0' ;
   }
   return ;
}
```

Annexe B

Fonctions de affiche_lib.c

- **LectureFITS** ouvre un fichier FITS, alloue un tableau aux bonnes dimensions pour stocker l'image, et renvoie l'adresse de ce tableau (sortie principale), les dimensions de cette image, ainsi que le nombre de secondes écoulées entre le 1er janvier 1970 et l'acquisition de l'image.

  PROTOTYPE:

```
long **LectureFITS ( char *filename,
                     long *dim1,
                     long *dim2,
                     long *nbsec )
```

  DESCRIPTION DES PARAMÈTRES:

filename	nom du fichier FITS à ouvrir	entrée
dim1	première dimension du cube (nombre de lignes)	sortie
dim2	deuxième dimension du cube (nombre de colonnes)	sortie
nbsec	nb de secondes entre le 01 / 01 / 1970 et l'acquisition de l'image	sortie

- **AfficheFITSEtLitPixel** affiche une image contenue dans un tableau et attend que l'on clique sur un pixel. Les coordonnées de ce pixel sont alors renvoyées.

  PROTOTYPE:

```
void AfficheFITSEtLitPixel ( long **cube,
                             long   dim1,
                             long   dim2,
                             int    img_min,
                             int    img_max,
                             int   *x,
                             int   *y )
```

  DESCRIPTION DES PARAMÈTRES:

`cube`	image à afficher	entrée
`dim1`	première dimension du cube (nombre de lignes)	entrée
`dim2`	deuxième dimension du cube (nombre de colonnes)	entrée
`img_min`	valeur de pixel à associer à la couleur noire	entrée
`img_max`	valeur de pixel à associer à la couleur blanche	entrée
`x`	coordonnée horizontale du pixel cliqué	sortie
`y`	coordonnée verticale du pixel cliqué	sortie

Annexe C

Librairies utilisées

C.1 Procédures de la librairie libcfitsio.a

- **fits_open_file** ouvre un fichier.

  PROTOTYPE:

```
int fits_open_file ( fitsfile **fptr,
                     char      *fichier,
                     int        rwmode,
                     int       *statut )
```

  DESCRIPTION DES PARAMÈTRES:

fptr	adresse du pointeur de type fitsfile avec lequel lier le fichier	entrée
fichier	nom du fichier à ouvrir	entrée
rwmode	mode d'ouverture du fichier. ouverture seule : READONLY ou ouverture/écriture : READWRITE.	entrée
statut	retourne 0 ou le code d'erreur en cas d'échec	sortie

- **fits_get_img_param** lit et retourne les valeurs des *keywords* standards.

  PROTOTYPE:

```
int fits_get_img_param ( fitsfile *fptr,
                         int       maxdim,
                         int      *bitpix,
                         int      *naxis,
                         long     *naxes,
                         int      *statut )
```

  DESCRIPTION DES PARAMÈTRES:

fptr	pointeur vers le fichier à lire	entrée
maxdim	nb maximal de dimensions des images	entrée
bitpix	nb de bits utilisés pour coder la valeur d'un pixel	sortie
nb_dim	nb de dimension de l'image	sortie
naxes	taille de chaque dimension	sortie
statut	retourne 0 ou le code d'erreur en cas d'échec	sortie

- **fits_read_pix** lit les pixels de l'image.

  PROTOTYPE:

```
int fits_read_pix ( fitsfile *fptr,
                    int       datatype,
                    long     *firstpixel,
                    long      nbpix,
                    void     *nullval,
                    void     *image,
                    int      *anynull,
                    int      *statut )
```

  DESCRIPTION DES PARAMÈTRES:

fptr	pointeur vers le fichier à lire	entrée
datatype	type de données à stocker (*i.e.* format du tableau image)	entrée
firstpixel	coordonnées du 1er pixel à lire	entrée
nbpix	nb de pixels à lire	entrée
nullval	valeur de remplacement pour les pixels non définis	entrée
image	tableau recevant les pixels lus	sortie
anynull	nombre de pixels non définis	sortie
statut	retourne 0 ou le code d'erreur en cas d'échec	sortie

- **fits_read_keyword** lit la valeur d'un mot-clé dans le *header*.

  PROTOTYPE:

```
int fits_read_keyword ( fitsfile *fptr,
                        char     *motcle,
                        char     *valeur,
                        char     *commentaire,
                        int      *statut )
```

  DESCRIPTION DES PARAMÈTRES:

fptr	pointeur vers le fichier à lire	entrée
motcle	mot-clé à lire	entrée
valeur	valeur associée au mot-clé ; c'est **toujours** une chaîne de caractères, même si sa valeur repésente un nombre	sortie
commentaire	commentaire associé au mot-clé	sortie
statut	retourne 0 ou le code d'erreur en cas d'échec	sortie

Dans tous les cas, la valeur de retour de ces fonctions est la même que la valeur retournée par la variable `statut`.

Le manuel exhaustif des procédures contenues dans cette librairies se trouve sur internet : http://heasarc.gsfc.nasa.gov/docs/software/fitsio/c/c_user/

C.2 Procédures de la librairie libX11.a

- **`XOpenDisplay`** ouvre une connexion avec le serveur X, et renvoie :
  - un pointeur sur une structure de type `Display` qui contient les informations sur le serveur X avec lequel la connexion a été établie. On se servira par la suite de ce pointeur pour envoyer les instructions au serveur.
  - `NULL` en cas d'échec.

  PROTOTYPE:

  ```
  Display *XOpenDisplay ( char *display_name )
  ```

  DESCRIPTION DES PARAMÈTRES:

`display_name`	Nom du serveur X à contacter. Pour se connecter au serveur par défaut de l'ordinateur, utiliser la valeur `NULL`	entrée

- **`DefaultScreen`** retourne le numéro de l'écran principal du serveur X (il peut y avoir plusieurs écrans connecté à un même ordinateur)

  PROTOTYPE:

  ```
  int XDefaultScreen ( Display *display )
  ```

  DESCRIPTION DES PARAMÈTRES:

`display`	pointeur vers le serveur X	entrée

- **`DefaultColormap`** retourne la table des couleurs utilisée par défaut par l'écran.

  PROTOTYPE:

  ```
  Colormap XDefaultColormap ( Display *display,
                              int      screen_number )
  ```

  DESCRIPTION DES PARAMÈTRES:

`display`	pointeur vers le serveur X	entrée
`screen_number`	numéro de l'écran	entrée

- **XCreateSimpleWindow** crée une fenêtre dont la description (position, taille, ...) est donnée par les paramètres de la fonction. La fenêtre n'est pas affichée. Elle ne l'est qu'à l'appel de la fonction `XMapWindow`. Retourne l'identifiant de la fenêtre créée.

  PROTOTYPE:

```
Window XCreateSimpleWindow ( Display      *display,
                             Window        parent,
                             int           x,
                             int           y,
                             unsigned int  width,
                             unsigned int  height,
                             unsigned int  border_width,
                             unsigned long border,
                             unsigned long background )
```

  DESCRIPTION DES PARAMÈTRES:

`display`	pointeur vers le serveur X	entrée
`parent`	numéro de la fenêtre parente. On utilise généralement `RootWindow(display, screen_num)` (où `screen_num` est le numéro de l'écran retourné par `DefaultScreen()`) pour spécifier que la fenêtre est une fenêtre principale et qu'elle n'a pas de parent	entrée
`x`	coordonnée horizontale du coin supérieur gauche de la fenêtre dans l'écran	entrée
`y`	coordonnée verticale du coin supérieur gauche de la fenêtre dans l'écran	entrée
`width`	largeur de la fenêtre	entrée
`height`	hauteur de la fenêtre	entrée
`border_width`	épaisseur en pixel du bord de la fenêtre	entrée
`border`	couleur du bord de la fenêtre	entrée
`background`	couleur de fond de la fenêtre	entrée

- **XMapWindow** affiche la fenêtre donnée en paramètre.

  PROTOTYPE:

```
int XMapWindow( Display *display,
                Window   w )
```

  DESCRIPTION DES PARAMÈTRES:

`display`	pointeur vers le serveur X	entrée
`w`	identifiant de la fenêtre à afficher	entrée

- **XSelectInput** sélectionne les événements (clavier et souris) pour lesquels le programme aura une réaction (par exemple, afficher les coordonnées d'un point de la fenêtre quand on clique dessus avec la souris).

PROTOTYPE:

```
int XSelectInput ( Display *display,
                   Window   w,
                   long     event_mask )
```

DESCRIPTION DES PARAMÈTRES:

display	pointeur vers le serveur X	entrée
w	identifiant de la fenêtre	entrée
event_mask	liste des évènements à traiter. Les valeurs possibles[1] sont * ButtonPressMask : le bouton de la souris est pressé * ButtonReleaseMask : le bouton de la souris est relâché * EnterWindowMask : la souris entre dans la fenêtre * LeaveWindowMask : la souris quitte la fenêtre * ExposureMask : la fenêtre est affichée (lors de la création ou lorsqu'une fenêtre qui la cachait a été déplacée) * KeyPressMask : une touche du clavier est pressée * KeyReleaseMask : une touche du clavier est relâchée * PointerMotionMask : la souris a bougé à l'intérieur de la fenêtre Pour combiner plusieurs évènements, on utilise l'opérateur « ou binaire » | .	entrée

- **XAllocColor** détermine le numéro de la couleur qui contient la valeur RGB demandée, ou qui contient la valeur RGB la plus proche disponible sur cet écran. La valeur RGB est stockée dans les champs red, green et blue de la variable de type XColor donnée en argument. Retourne 0 en cas d'erreur.

  PROTOTYPE:

  ```
  Status XAllocColor ( Display  *display,
                       Colormap  colormap,
                       XColor   *color_in_out )
  ```

  DESCRIPTION DES PARAMÈTRES:

1. Voir http://tronche.com/gui/x/xlib/events/processing-overview.html pour la liste complète.

`display`	pointeur vers le serveur X	entrée
`colormap`	table de couleurs à utiliser	entrée
`color_in_out`	structure de type `XColor` contenant la valeur (champs `red`, `green` et `blue`) de la couleur demandée. En sortie, le champ `pixel` contient le numéro de la couleur dans la table de couleurs.	entrée / sortie

- **`DefaultGC`** retourne le contexte graphique (couleur de fond, couleur de dessin, fonte, etc.) associé à l'écran donné en paramètre.

  PROTOTYPE:

```
GC XDefaultGC ( Display *display,
                int      screen_number )
```

  DESCRIPTION DES PARAMÈTRES:

`display`	pointeur vers le serveur X	entrée
`screen_number`	numéro de l'écran	entrée

- **`XSetForeground`** change la couleur de dessin.

  PROTOTYPE:

```
int XSetForeground(Display        *display,
                   GC              gc,
                   unsigned long   foreground )
```

  DESCRIPTION DES PARAMÈTRES:

`display`	pointeur vers le serveur X	entrée
`gc`	contexte graphique à modifier	entrée
`foreground`	numéro de la couleur dans la table de couleurs (champ `pixel` d'un `XColor`)	entrée

- **`XDrawPoint`** dessine un point aux coordonnées `(x,y)` en utilisant la couleur définie dans le contexte graphique (`foreground`).

  PROTOTYPE:

```
XDrawPoint ( Display  *display,
             Drawable  d,
             GC        gc,
             int       x,
             int       y )
```

  DESCRIPTION DES PARAMÈTRES:

display	pointeur vers le serveur X	entrée
d	identifiant de l'objet dans lequel dessiner. Quand on veut dessiner dans une fenêtre, c'est l'identifiant de la fenêtre	entrée
gc	contexte graphique à utiliser	entrée
x	coordonnée horizontale du point	entrée
y	coordonnée verticale du point	entrée

- **XNextEvent** retourne le prochain évènement (souris ou clavier) à traiter. Seuls les évènements qui sont passés au travers du crière défini par XSelectInput sont traités.

  PROTOTYPE:

```
XNextEvent ( Display *display,
             XEvent  *event_return )
```

  DESCRIPTION DES PARAMÈTRES:

display	pointeur vers le serveur X	entrée
event_return	prochain évènement à traiter	sortie

- **XKeycodeToKeysym** retourne le symbole associé au code de touche donné en argument. Selon le clavier utilisé, le code d'une touche ne correspond pas toujours au même caractère. Chaque caractère est associé à un symbole (par exemple, la lettre q a pour symbole XK_Q, le caractère : a pour symbole XK_semicolon)

  PROTOTYPE:

```
KeySym XKeycodeToKeysym ( Display *display,
                          KeyCode  keycode,
                          int      index )
```

  DESCRIPTION DES PARAMÈTRES:

display	pointeur vers le serveur X	entrée
keycode	code de la touche pressée. On le trouve dans event.xkey.keycode	entrée
index	Une touche peut avoir plusieurs significations (majuscule / minuscule, chiffres / symboles). Cet index indique le numéro de la signification. 0 correspond à la touche pressée seule. D'autres index indiquent que la touche a été pressée avec <shift>, <ctrl>, etc.	entrée

- **XKeysymToString** retourne une chaîne de caractère contenant la lettre corrspondant au symbole donné en argument.

  PROTOTYPE:

```
char *XKeysymToString ( KeySym keysym )
```

DESCRIPTION DES PARAMÈTRES:

`keysym`	symbole de caractère	entrée

item **`XCloseDisplay`** ferme la connexion vers le serveur X.

PROTOTYPE:

```
XCloseDisplay ( Display *display )
```

DESCRIPTION DES PARAMÈTRES:

`display`	pointeur vers le serveur X	entrée