

Introduction à l'imagerie numérique

3I022-fev2018

Présentation du logiciel INRIMAGE

Licence d'informatique



Février 2018

Plan

Introduction

Le format Inrimage

Commandes Inrimage

Visualisation des images

Bibliothèque de développement

Conclusion sur Inrimage

Histoire du logiciel

- ▶ Logiciel INRIA
- ▶ Développement initial date des années 80 (vieux logiciel) en Fortran pour le besoin en recherche d'équipes INRIA.
- ▶ Conçu pour tourner sur des architectures très diverses (Sun,Dec,PC) voire même exotiques (Cray 64 bits et 128 bits).
- ▶ Passage en C au début des années 90, la moitié du code reste en Fortran.
- ▶ Rendu public vers 1998.

Aujourd'hui

- ▶ Site officiel <http://inrimage.gforge.inria.fr>
- ▶ Support sur PC/Linux 32 et 64 bits
- ▶ Distribution de type GNU, autotools depuis 2006.
- ▶ Packaging fait pour Ubuntu (Debian), Fedora (Redhat), Cygwin (Windows), Mac OSX (Homebrew)

Installation

- ▶ Les distributions sont disponibles ici¹ :

```
|| depot=http://inrimage.gforge.inria.fr/dist/
```

- ▶ Debian (ou Ubuntu au même endroit) 64 bits :

```
|| wget $depot/4.6.8/inrimage_4.6.8-jessie_amd64.deb  
|| sudo dpkg -i inrimage_4.6.8-jessie_amd64.deb  
|| sudo aptitude install inrimage
```

Remarque : il faut ignorer les messages d'avertissement de **dpkg**.

- ▶ Fedora core 24, 64 bits :

```
|| su  
|| rpm -i $depot/4.6.8/inrimage-4.6.8-static_R1.x86_64.rpm
```

1. Dans ce cours, les interactions avec le shell commencent par une double barre

Installation (suite)

- ▶ Windows : on peut utiliser Cygwin, **non recommandé** !
Installation de Cygwin : voir
<http://pequan.lip6.fr/~bereziat/inrimage/cygwin>

```
|| cd /
|| curl $depot/4.6.5/inrimage-4.6.5-cygwin64.tgz | tar xz
```

- ▶ Windows, **recommandé** : utiliser une machine virtuelle (Virtual Box, libre) et la distribution Debian (Jessie) utilisée à la PPTI.

Suivre les directives de

<http://pequan.lip6.fr/~bereziat/envdev/>, section
“Virtual Box”, ignorez le point 6.

- ▶ Mac OSX : utiliser Virtual Box, ou bien, directement dans Mac OSX :

1. Installer Homebrew : <https://brew.sh>
2. Puis :

```
|| brew tap bereziat/inrimage
|| brew install inrimage
```

Installation (fin)

- ▶ Autre distribution Linux, BSD, ... : par compilation du logiciel !
- ▶ Installer les dépendances obligatoires : gfortran, Xaw ou Xaw3d
- ▶ Installer les dépendances recommandées : libjpeg, libpng, libtiff, netpbm
- ▶ Compilation :

```
wget $depot/4.6.8/inrimage-4.6.8.tar.gz
tar xvfz inrimage-4.6.8.tar.gz
cd inrimage-4.6.8
./configure
make
su
make install
```

Structure du logiciel

- ▶ Des bibliothèques de développement :
 1. une bibliothèque principale (documentée) gérant les entrées/sorties et les interactions avec l'utilisateur
 2. une bibliothèque de gestion des fichiers de contours (partiellement documentée)
 3. une bibliothèque d'interfacage avec Fortran
 4. une bibliothèque d'affichage sous GTK2 (documentée)
 5. des bibliothèques de fonctions de traitement d'images (malheureusement non documentées) : elles correspondent aux fonctionnalités implantées dans les commandes Inrimage.
- ▶ Des commandes, respectant les conventions d'Unix, à utiliser par l'intermédiaire d'un interpréteur de commandes. Ces commandes sont construites à partir des bibliothèques de développement.
- ▶ Une documentation au format man et html.
- ▶ Quelques exemples d'utilisation.

Un format d'image propre

- ▶ Format INRIMAGE : plusieurs évolutions au cours de son histoire.
- ▶ La dernière version (4), est stable depuis 1990. Tous les formats antérieurs sont néanmoins reconnus par le logiciel.
- ▶ Pourquoi un format propre ?
 - ▶ répondre aux besoins scientifiques des équipes INRIA de l'époque (les formats d'alors ne convenaient).
 - ▶ le traitement scientifique des images nécessite souvent un codage à virgule flottant, codage généralement inexistant dans les autres formats d'images.
 - ▶ Liste assez exhaustive de formats, voir :
<https://www.imagemagick.org/script/formats.php>

Un format d'image propre

suite

- ▶ Principales caractéristiques du format INRIMAGE :
 - ▶ codage des images en virgule fixe ou flottante ;
 - ▶ en flottant : simple et double précision sont possible ;
 - ▶ en virgule fixe : le codage à un nombre de bit arbitraire est possible, on peut signer le codage ;
 - ▶ une image est structurée en plan et les pixels en un nombre quelconque de composantes. Ceci permet de gérer des formats 4D d'image ;
 - ▶ portabilité entre différentes architectures (pas le même format de flottant d'une archi à l'autre, gestion de l'*endianess*) ;
 - ▶ table des couleurs ;
 - ▶ commentaires et historique des commandes utilisées pour créer l'image ;

D'autres formats d'image

- ▶ Inrimage ne sait manipuler que des images INRIMAGE, on l'a dit.
- ▶ Pour lire d'autre type d'image : il faut convertir !

Exemple

Pour lire du JPEG, convertir vers Inrimage avec la commande jpg2inr.

- ▶ Ceci est fastidieux : Inrimage utilise un fichier de configuration qui lui indique quelle commande utiliser lorsque tel type d'image est rencontré :
 - ▶ Le type d'image est donné soit par son extension soit par son *numéro magique* (à la façon de `/etc/magic` et de la commande `file`).
 - ▶ Lorsqu'on tente d'ouvrir une image non INRIMAGE, Inrimage cherche une entrée dans ce fichier, et si il la trouve, utilise le filtre mentionné pour convertir l'image. Ceci est transparent pour l'utilisateur.

Écriture vers d'autres formats d'image

- ▶ Inrimage ne sait qu'écrire à son format ; mais on peut utiliser les filtres de conversion disponibles.

Exemple

Pour écrire du JPEG, convertir avec la commande `inr2jpg`.

- ▶ Comme les commandes Inrimage peuvent fonctionner en pipeline, il suffit d'ajouter le filtre de conversion souhaité en fin de pipeline.

Exemple (Égalisation d'histogramme d'une image jpeg)

```
||      eg lena.jpg | inr2jpg >lena_eq.jpg
```

Autres formats

- ▶ Formats supportés :
 - ▶ Lecture et écriture : JPEG, GIF89, PNG, BMP, PGM, PBM, PNM, JPEG2000, TIF, XBM.
 - ▶ Lecture seule : Sun rasterfile, MPEG.
- ▶ Encore d'autres formats ? en utilisant d'autres logiciels de manipulation d'images (Netpbm, ImageMagick, ...) : il est facile d'accéder aux formats non supportés par Inrimage en combinant les bons filtres.

Autres logiciels ?

- ▶ Pour le traitement **scientifique** des images, liste non exhaustive de logiciels utilisés dans la communauté du traitement des images.
 - ▶ Matlab/Octave/IDL : programmation matricielle/vectoriel très bien adaptée au traitement d'images.
 - ▶ Python+Numpy+Matplotlib+Pillow : langage de script généraliste orienté objet (le logiciel qui monte, notamment en Machine Learning).
 - ▶ ImageJ, Java2D : nécessite JAVA.
 - ▶ Bibliothèque C/C++/Python : OpenCV (<http://opencv.org>).
 - ▶ Des logiciels très spécialisés et chers (ex : en imagerie médicale, satellitaire).

Autres logiciels (suite) ?

- ▶ Positionnement d'Inrimage par rapport à ces logiciels :
 - ▶ Inrimage est bas-niveau et spécialisé pour l'utilisation en script.
 - ▶ Les manipulations sur les formats et structures des images en tant que fichier sont aisées et efficaces.
 - ▶ On a une grande portabilité d'un système à l'autre.
 - ▶ Grande variété de codage d'image possible.
 - ▶ Langage C, C++ ou Fortran. Pas besoin d'apprendre un nouveau langage.
- ▶ Ce que Inrimage n'a pas : des algorithmes haut-niveau de traitement d'images. Il suffit d'utiliser OpenCV !

Plan

Introduction

Le format Inrimage

Commandes Inrimage

Visualisation des images

Bibliothèque de développement

Conclusion sur Inrimage

Structure du format Inrimage (version 4)

- ▶ Une entête en ASCII décrivant les caractéristiques de l'image :
 - ▶ l'entête a une longueur multiple de 256 octets :

```
par lena.inr -hdr && echo  
-hdr=1  
cat lena.inr | head -c 255 | sed '/^$/d'  
#INRIMAGE-4#{  
XDIM=256  
YDIM=256  
ZDIM=1  
VDIM=1  
TYPE=unsigned fixed  
PIXSIZE=8 bits  
SCALE=2**0  
CPU=decm  
#[H]*<1> izoom lena-bw.inr.gz lena-256x256.inr -x 256 -y  
##}
```

Structure du format Inrimage

suite

- ▶ On peut mettre toute sorte d'information (y compris les siennes). Pour coder une table des couleurs, il faut une grande entête :

```
par selection.inr.gz -hdr && echo  
-hdr=8  
zcat selection.inr.gz | head -c $((256*8))  
#INRIMAGE-4#{  
XDIM=256  
YDIM=256  
ZDIM=1  
VDIM=1  
TYPE=unsigned fixed  
PIXSIZE=8 bits  
SCALE=2**0  
CPU=decm  
COLTBR=A(0-255)12597E8CB1883169C8AEC19342818D4EB58DDE83BF4987B2C9BD84  
COLTBG=A(0-255)0387C20C884746A646A9C765658A806A9278AC6CC286A8A67298A  
COLTBB=A(0-255)0432630E371C0D492B445817273249068D134798AD0B4973302263  
#[H]*<1> gif2inr selection.gif selection.inr
```

Structure du format Inrimage

suite

- ▶ Après l'entête, on trouve les valeurs des pixels stockées contiguement, en binaire, dont l'organisation est décrite à la fin de ce cours (voir Tableau de format).
- ▶ Le format Inrimage n'est pas compressé. Toutefois les images peuvent être compressées : Inrimage reconnaît nativement le format `gzip` mais d'autres décompresseurs peuvent facilement être reconnus (voir Configuration d'Inrimage).

Format d'une image

- ▶ **Format=Codage+Dimensions** : les caractéristiques structurelles d'une image.
- ▶ **Codage** : comment est codé numériquement un pixel.
 - ▶ type de codage : flottant, virgule fixe,
 - ▶ taille du codage : en octet ou en bit,
 - ▶ si codage à virgule fixe : exposant.
- ▶ **Dimensions** : comment sont organisés les pixels dans l'image.
 - ▶ composantes par pixel ?
 - ▶ pixels par ligne ?
 - ▶ lignes par plan ?
 - ▶ plans ?
- ▶ **Format étendu** : format + autres informations plus ou moins utiles (biais, facteur d'échelle, type de maille, ...).
- ▶ Ces informations sont données par la commande `par`.

Format : la commande **par**

Exemple (Format d'une image)

```
|| par lena.inr  
|| lena.inr      -F=Inrimage -hdr=1    -x 256   -y 256   -f   -o 1
```

-x	-y	-z	-v	dimensions de l'image
-o	-b			taille de codage en octet, en bit
-f	-r			virgule fixe, flottante
	-F=			format de l'image
	-hdr=			taille de l'entête

Exercice

Avec la commande :

```
|| man par
```

*lire le manuel de la commande **par**.*

Plan

Introduction

Le format Inrimage

Commandes Inrimage

Visualisation des images

Bibliothèque de développement

Conclusion sur Inrimage

Paramétrisation d'une commande

- ▶ De façon conventionnelle une commande Inrimage accepte deux type de paramètres : des noms d'images et des options.
- ▶ Les noms d'images indiquent, selon les commandes, les images qui doivent lues et celles qui doivent être écrites.
 - ▶ L'ordre est important : d'abord les entrées puis les sorties.
 - ▶ Un nom d'image ne doit pas commencer par un chiffre.
 - ▶ Si un nom d'image est omis, alors la commande lit dans l'entrée standard ou écrit dans la sortie standard. On peut utiliser le caractère '-' lorsque c'est ambiguë.

Exemple

- ▶ La commande *ad* ajoute deux images :

|| **ad** *im1 im2 im1plus2*

- ▶ On peut ajouter trois images en combinant les appels à *ad* (on notera ici l'usage de '-' pour lever l'ambiguïté).

|| **ad** *im1 im2 / ad - im3 im1plus2plus3*

Paramétrisation des commandes (suite)

- ▶ La plupart des commandes Inrimage utilisent le format (taille et codage) de l'image de sortie si elle existe.
- ▶ Par conséquence : indiquer un nom d'image sortie ou rediriger la sortie ne sont généralement pas équivalents :

Exemple

- ▶ *la commande ad écrit toujours en codage flottant,*
- ▶ *la commande cco effectue une conversion de codage.*

```
par imlplus2
imlplus2 -F=Inrimage -hdr=1 -x 256 -y 256 -r -o 4
ad im1 im2 | cco -f - imlplus2
par imlplus2
imlplus2 -F=Inrimage -hdr=1 -x 256 -y 256 -r -o 4
ad im1 im2 | cco -f > imlplus2
par imlplus2
imlplus2 -F=Inrimage -hdr=1 -x 256 -y 256 -f -o 1
```

Paramétrisation des commandes (suite)

- ▶ Les options : ce sont des mots qui commencent par - et qui peuvent être suivi d'une, deux ou aucune valeur numérique.
- ▶ L'ordre des options n'est pas signifiant.
- ▶ Les valeurs numériques doivent impérativement suivre l'option.
- ▶ Lorsque certaines options essentielles manquent, la commande peut les demander en mode *interactif*. Exemple :

```
|| raz  
|| dimx, dimy, dimv, dimz (0 eq 1) _
```

- ▶ Certaines commandes (en nombre réduit) sont purement interactives.
- ▶ Il est facile (avec le shell), de les rendre non interactive (voir annexe).

Options communes

- ▶ Les commandes Inrimage possèdent des options communes.
- ▶ Elles modifient le comportement par défaut des commandes Inrimage.
- ▶ Certaines de ces options sont également doublées par des variables d'environnement.

Option	Description	Variable env.
-vers	affiche dans stderr le numéro de version	
-help	affiche dans stderr l'aide de la commande	
-Help	affiche dans stderr l'aide abrégé	
-D	mode debug avec tracage sur stderr	
-.ext	force le type des images lues	
-F=type	les images sont créées avec le type 'type'	IMAGE_OUT=type
-hdr=nn	les images sont créées avec un entête de nn*256 octets	IMAGE_HDRL=nn
-k nn	indique la taille de mémoire (en k-octets) que la fonction imexec() peut utiliser	INR_KBMEM=nn

Options communes (suite)

- ▶ Les options `-.ext` et `-hdr` sont parfois utiles dans certaines situations. L'option `-k` est pour les experts.

Exemple

```
file /Cache/E39A1965d01
Cache/E39A1965d01 PNG image, 383 x 303, 8-bit/color RGBA
par - .ext png Cache/E39A1965d01
Cache/E39A1965d01      -F=Inrimage    -hdr=1 -x 383   -y 303
-v 4   -f   -o 1
```

- ▶ **Attention** : le nom des options communes est réservé.

Plan

Introduction

Le format Inrimage

Commandes Inrimage

Visualisation des images

Bibliothèque de développement

Conclusion sur Inrimage

La commande xvis

- ▶ Les images sont visualisées avec la commande xvis.
- ▶ Interface austère mais commande puissante.
- ▶ Principales fonctionnalités :
 - ▶ affichage d'un ensemble d'images avec possibilité de groupement et d'animation
 - ▶ animation sur les plans et/ou les composantes d'une image
 - ▶ sur-échantillonnage (zoom) ou sous-échantillonnage
 - ▶ communication avec l'extérieur par réception de signaux
 - ▶ mode d'édition (simpliste) d'image :
 - ▶ point, ligne, remplissage de rectangle, copie de zones
 - ▶ copie d'écran
 - ▶ superposition de deux images
 - ▶ manipulation de la table des couleurs
 - ▶ interface avec le shell : on peut déclencher des scripts depuis xvis, interagir avec le shell. cela permet de créer des démonstrateurs à peu de frais.

Démonstration d'xvis

- ▶ Quelques fonctionnalités qui vous seront utiles en TME.

|| **xvis** lena.inr



Démonstration d'xvis (suite)

- ▶ Affichage réduit :

```
|| xvis -nu lena.inr
```



Démonstration d'xvis (suite)

- ▶ Affichage groupé verticalement :

```
|| xvis -nu lena.inr cameraman.inr -grp 2
```



Démonstration d'xvis (suite)

- ▶ Affichage groupé horizontalement :

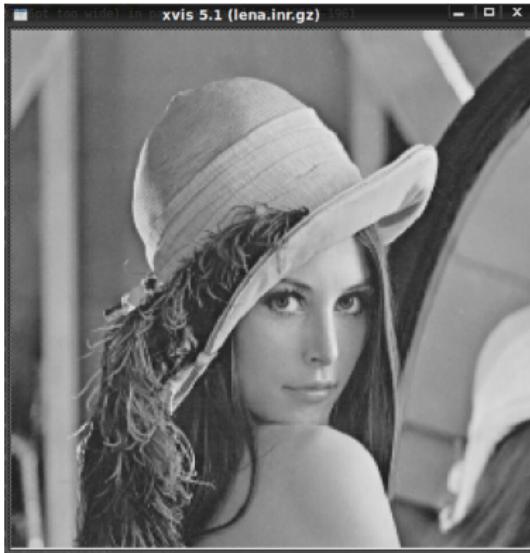
```
|| xvis -nu lena.inr cameraman.inr -grp 2 -hz 2
```



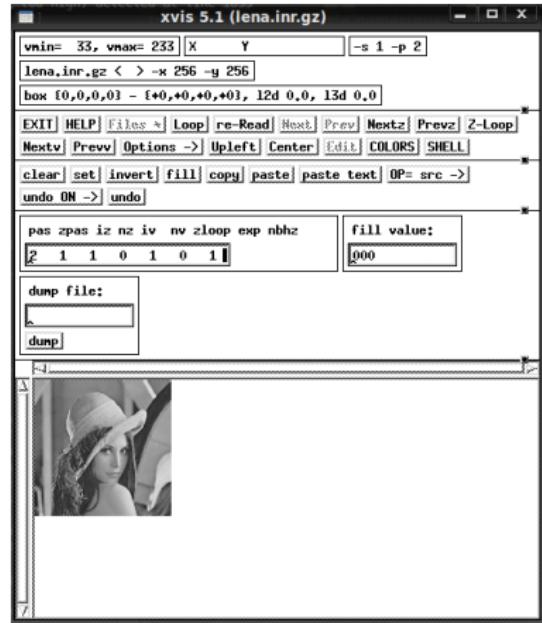
Démonstration d'xvis (suite)

- ▶ Sur ou sous-échantillonnage :

```
xvis -nu -p -4 lena.inr
```



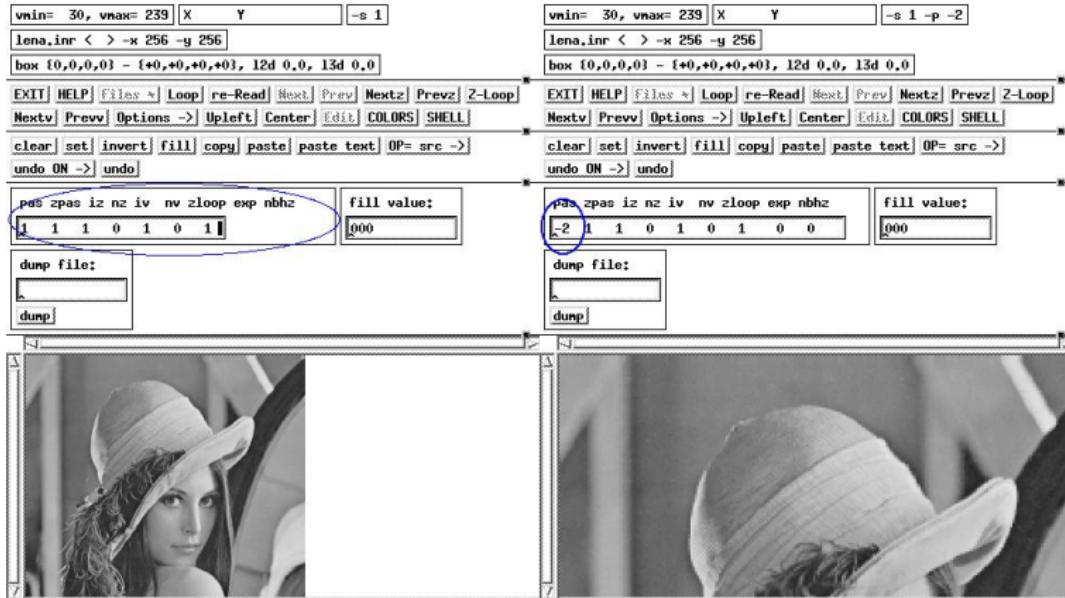
```
xvis -p 2 lena.inr
```



Démonstration d'xvis (suite)

Interaction

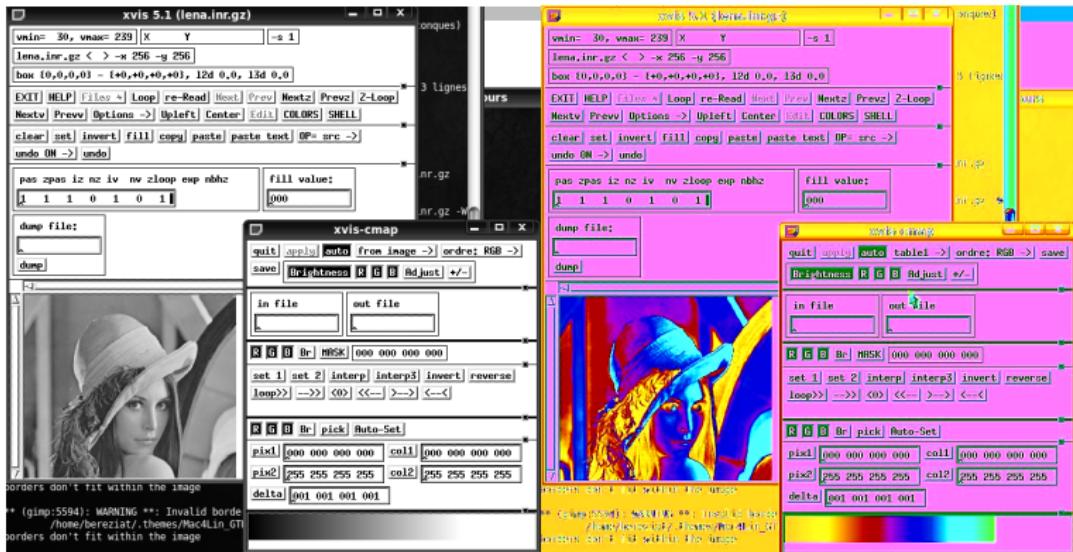
- Certains paramètres peuvent être modifiés en cours d'exécution (éditer le paramètre puis appuyer sur 'n') :



Démonstration d'xvis (suite)

- Opérations complexes sur la table des couleurs

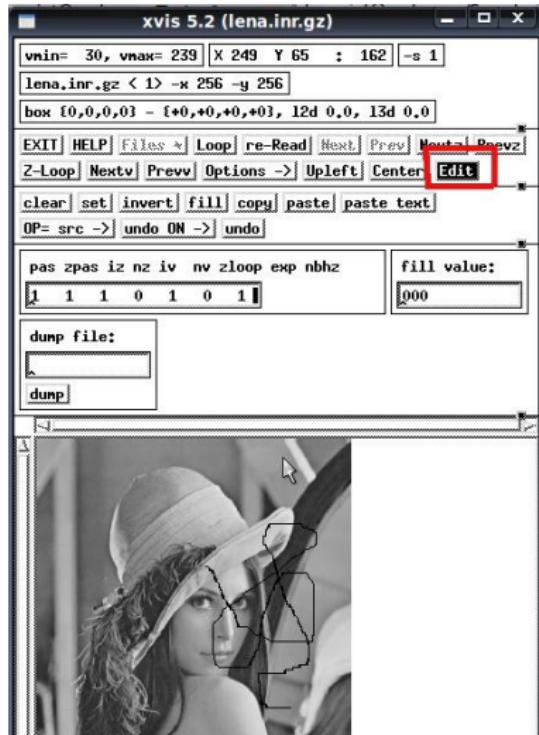
|| **xvis** lena.inr -Wc



- Changement de table des couleurs (affecte la table d'X11 pas l'image).

Démonstration d'xvis (suite)

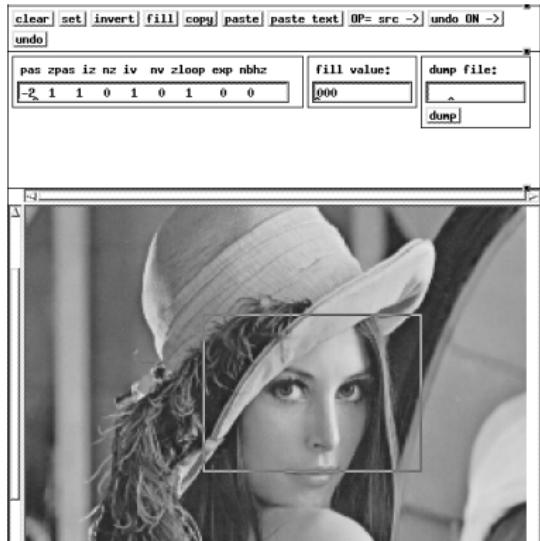
Édition



- ▶ Mode (primitif) d'édition des images :
 - || **raz** ` **par** -x -y lena.inr` mask
 - || **xvis** -ed lena.inr mask
- ▶ Clic droit : dessiner un point ou à main levée
- ▶ Clic milieu : dessiner un segment depuis le point précédent
- ▶ Shift-clic milieu : dessiner une boîte depuis le point précédent
- ▶ Seule la seconde image est modifiée. On peut spécifier deux fois la même image.

Démonstration d'xvis (suite)

Région d'intérêt



- ▶ Sélection d'une région :
 - ▶ Shift-clic milieu : sélection du premier coin,
 - ▶ shift-clic gauche : sélection du second coin,
 - ▶ Shift-clic droit : activation/désactivation de la fenêtre.
- ▶ Opérations simples sur la région :
 - ▶ initialisation,
 - ▶ inversion,
 - ▶ copier-coller,
 - ▶ insertion de texte.
- ▶ Prise en compte par les scripts...

Démonstration d'xvis (suite)

Région d'intérêt : quelques opérations



Démonstration d'xvis (suite)

Interface avec l'interprète de commande

```
|| xvis lena.inr -Wsh -nu
```

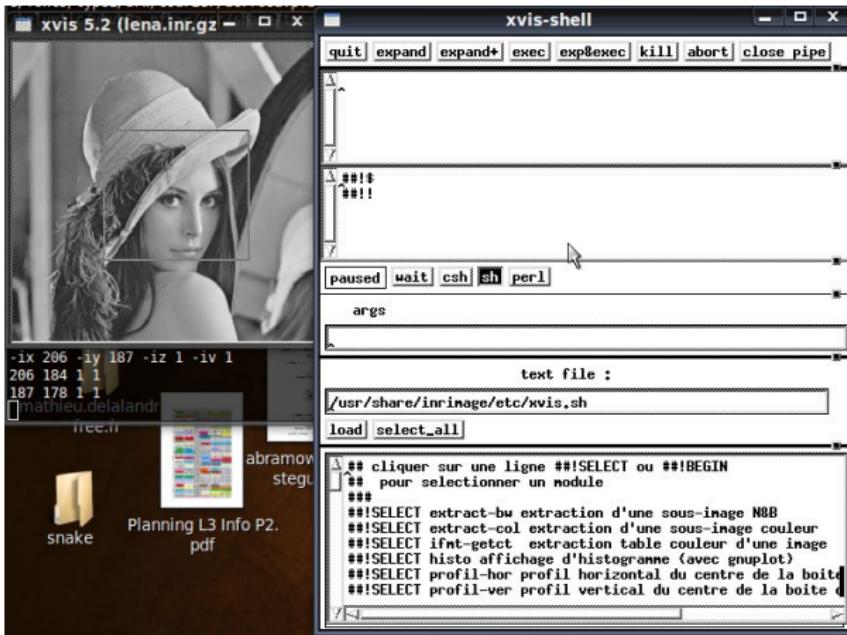


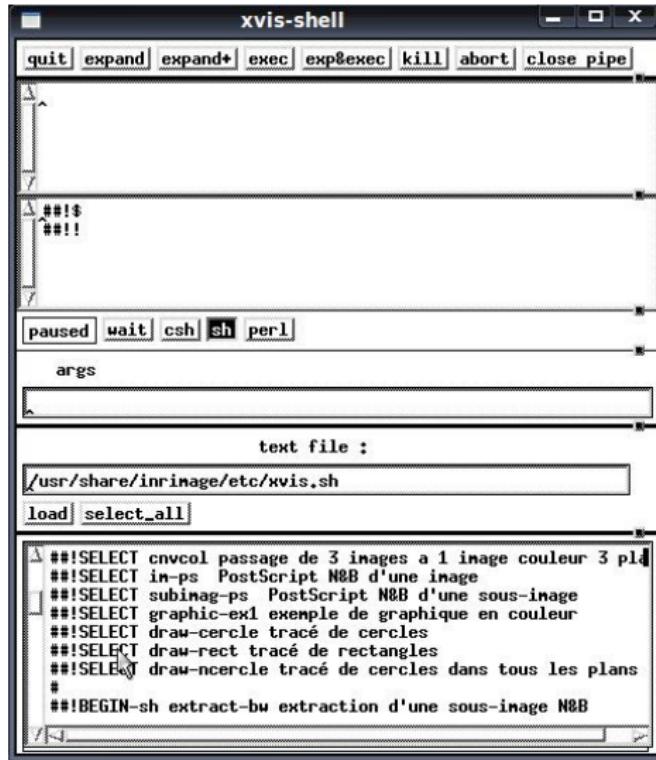
FIGURE – Lancement avec ouverture de la fenêtre shell et choix d'une

Introduction à l'imagerie numérique -

Licence d'informatique - SU

Interface avec l'interprète de commande

Scripts par défaut et menu



Interface avec l'interprète de commande

Choix du script : ici dessin de cercle

The screenshot shows the xvis-shell window with the following interface elements:

- Top Bar:** A menu bar with the title "xvis-shell" and several buttons: quit, expand, expand+, exec, exp&exec, kill, abort, close pipe.
- Script Editor:** A large text area containing a script:

```
#!/BEGIN draw-cercle tracé de cercle dans la boîte
##!nobox
##!draw(z $z0,a $x0 $y0 $dx $dy)
```
- Tool Buttons:** A row of buttons: paused, wait, csh, sh, perl.
- Arguments:** A text input field labeled "args".
- Text File:** A text input field labeled "text file :" containing the path "/usr/share/inrimage/etc/xvis.sh".
- Action Buttons:** A row of buttons: load, select_all.
- Bottom Terminal:** A terminal window showing the output of the command "xvisdraw -c 130 <<%". The output includes coordinates (1, 1), (20, 1), (20, 20), (1, 20), (1, 1), and a final "%". Below this, the script is repeated:

```
#!/BEGIN draw-cercle tracé de cercle dans la boîte
##!nobox
```

Interface avec l'interprète de commande

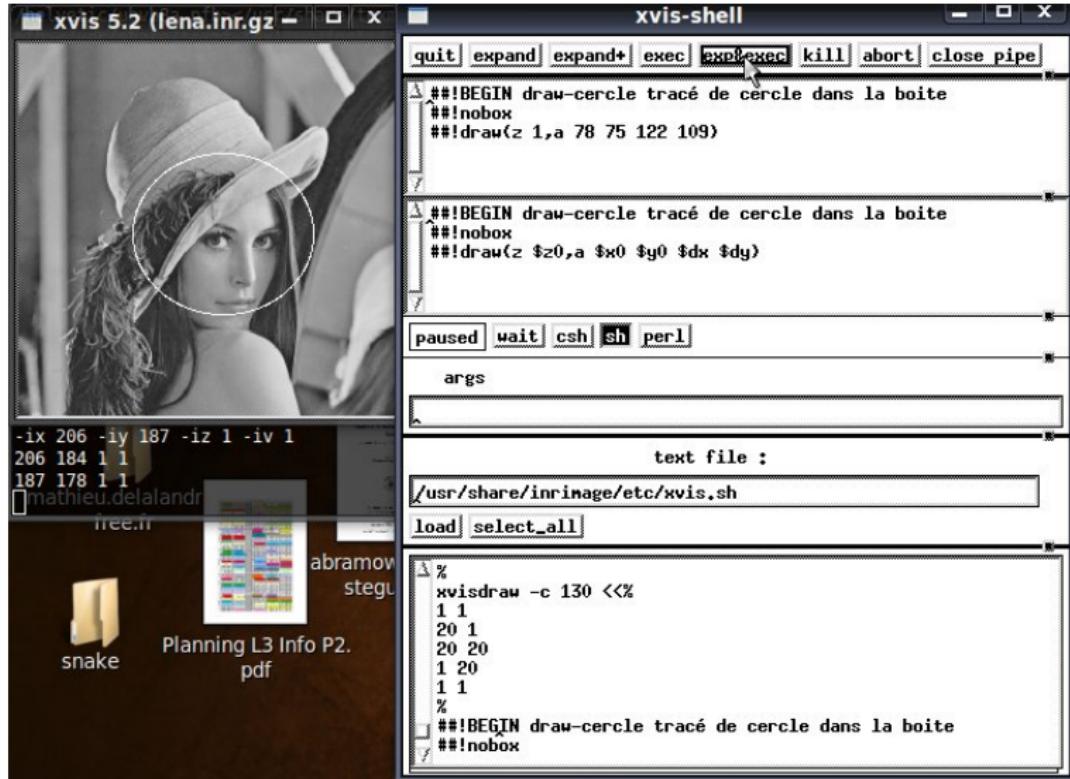
Substitution des variables puis exécution

The screenshot shows an xvis-shell window with the following interface elements:

- Top Bar:** quit, expand, expand+, exec, exec&exec, kill, abort, close pipe.
- Text Area:** Displays command history and output. The first two lines are comments starting with '##!BEGIN'. The third line is a command with variables \$z, \$a, \$x0, \$y0, \$dx, and \$dy. The fourth line is another comment starting with '##!BEGIN'. The fifth line is a command with variables \$z0, \$a, \$x0, \$y0, \$dx, and \$dy. The sixth line is a command starting with 'xvisdraw -c 130 <<%'. The seventh line is a command starting with '%'. The eighth line is a comment starting with '##!BEGIN'.
- Buttons Bar:** paused, wait, csh, sh, perl.
- Args Bar:** An empty input field labeled "args".
- Text File Input:** A text file input field containing the path "/usr/share/inrimage/etc/xvis.sh".
- Buttons Bar:** load, select_all.
- Bottom Text Area:** Displays the output of the commands entered, including the substitution of variables and the final command 'xvisdraw -c 130 <<%'.

Démonstration d'xvis (suite)

Interface avec l'interpréte de commande



Scripts xvis

- ▶ Des variables propres à xvis : elles sont évaluées avant l'exécution du script par le shell (choix du shell). Quelques exemples :
 - ▶ \$im nom de l'image,
 - ▶ \$ndimx nombre de colonnes dans l'image,
 - ▶ \$x1 abscisse du coin bas droit de la dernière boite définie.
- Voir le manuel d'xvis, paragraphe LISTE DES VARIABLES
(man xvis, puis taper /LISTE DES VAR [RET])
- ▶ Des commandes xvis : ce sont des séquences commençant par ##!. Ce sont des commandes qui correspondent à l'action d'un bouton ou des primitives de dessin.
 - ▶ Par exemple ##!dump provoque une copie d'écran (appui sur le bouton dump).
- Voir le manuel d'xvis, paragraphe LISTE DES COMMANDES INTERNES (man xvis, puis taper /LISTE DES COM [RET])

Primitives de dessin

- ▶ Nous nous servirons des scripts essentiellement pour générer des primitives de dessin dans nos images.
- ▶ La primitive `draw` dessine sur l'image mais ne modifie pas l'image (les coordonnées de pixels commencent à 1) :

```
cat > lignes <<EOF
##!draw(C 255)
##!draw(p 10 10)
##!draw(l 90 90)
##!draw(p 10 90, l 90 10)
EOF
xvis -fnull 100 100 -xsh lignes
```

- ▶ Primitives disponibles :
 - ▶ point (p), ligne (l), ligne brisée (L),
 - ▶ rectangle (r), arc d'ellipse (a),
 - ▶ choix de la couleur (C), du plan (z)

Édition d'image

- ▶ Possibilité de sauver l'image en utilisant le *dump* :

```
cat > lignes <<EOF
##!draw(C 255)
##!draw(p 10 10)
##!draw(l 90 90)
##!draw(p 10 90,l 90 10)
##!dump(lignes.inr)
##!EXIT
EOF
xvis -fnull 100 100 -xsh lignes
xvis lignes.inr
```

Édition d'image

suite

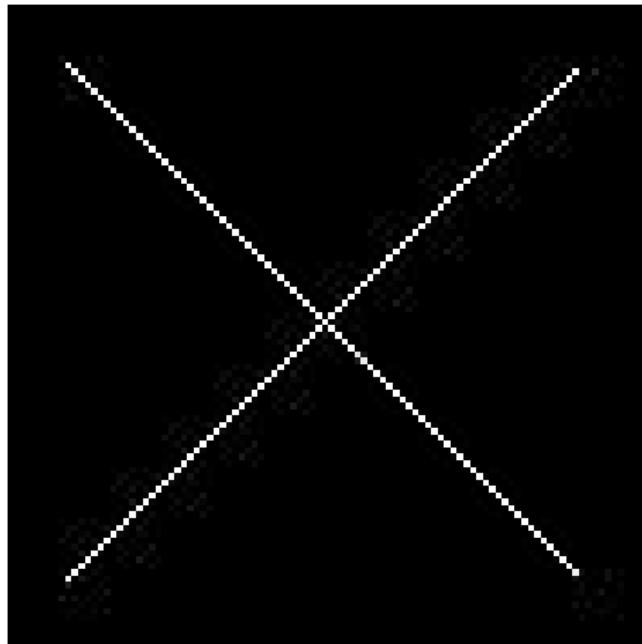
- ▶ La primitive d'édition `edit` modifie directement le contenu de l'image et nécessite d'utiliser `xvis` en mode d'*édition* :

```
cat > lignes <<EOF
##!edit (on)
##!edit (C 255)
##!edit (p 10 10,1 90 90)
##!edit (p 10 90,1 90 10)
##!EXIT
EOF
raz -x 100 -y 100 > lignes.inr
xvis -ed lignes.inr lignes.inr -xsh lignes
```

- ▶ Primitive plus restreinte :
 - ▶ Choix de la couleur (C), du plan (z),
 - ▶ point (p) et ligne (l),
 - ▶ la sous-primitive `on` correspond à l'appui sur le bouton Edit (passage en mode d'édition d'`xvis`)

Dessiner dans les images

- ▶ Les deux exemples précédents produisent donc l'image suivante :



La commande `gvis`

- ▶ La commande `gvis` est un visualisateur primitif (en comparaison avec `xvis`) écrit en GTK. Il permet de remplacer `xvis` lorsque celui ci est défaillant sur certains affichages (notamment en 16 bits) mais ces cas sont de plus en plus rares.
- ▶ La commande repose sur la bibliothèque `libgvis`. Cette dernière offre la possibilité d'afficher une image (de type quelconque) dans un *widget* GTK ce qui est utile pour réaliser des petits programmes de démonstration avec une interface graphique facile à programmer.

```
|| man libgvis
```

```
|| cd $INR_HOME/share/inrimage/gvis-demo
```

Autres commandes de visualisation

- ▶ Visualisation des valeurs de l'image : commande `tpr`

Exemple

```
tpr lena.inr | head -4  
ligne 1  
165 165 164 162 165 164 160 163 159 164  
157 159 159 159 157 158 157 159 161 162  
169 168 174 176 173 172 173 166 154 158
```

On peut contrôler le format d'affichage des valeurs :

```
tpr lena.inr -f=%f -c -l 6 | head -4  
0.647059 0.647059 0.643137 0.635294 0.647059 0.643137  
0.627451 0.639216 0.623529 0.643137 0.615686 0.623529  
0.623529 0.623529 0.615686 0.619608 0.615686 0.623529  
0.631373 0.635294 0.662745 0.658824 0.682353 0.690196
```

Graphiques PostScripts

- ▶ Des commandes pour fabriquer des graphiques à partir d'images :
 - ▶ `cou` : pour afficher des coupes d'images (profil)
 - ▶ `cour` : pour afficher des courbes d'image mono dimensionnel (utile couplé avec his, par exemple)
 - ▶ `pl` : pour afficher une image 2D sous la forme d'un graphe de surface.
- ▶ Ces commandes utilisent une variable d'environnement (`TEK_NOM`) pour déterminer de type de graphisme à produire.
- ▶ Graphique de type Tektronik (format obsolète)
- ▶ Graphique de type PostScript : `TEK_NOM=%`
- ▶ Deux façons d'appeler les commandes :

```
|| export TEK_NOM=%
```

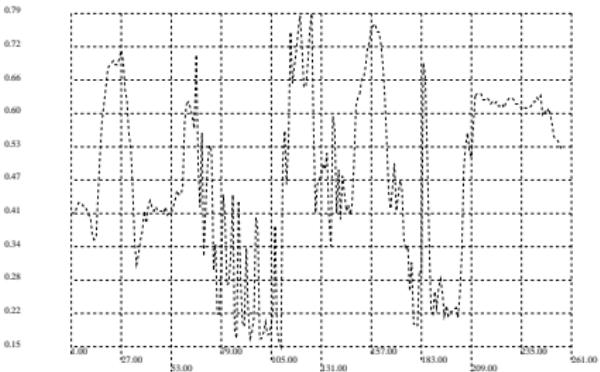
```
|| pl image | gv -
```

```
|| TEK_NOM=% pl image | gv -
```

Exemples

Exemple (Coupes dans les images)

```
extg ../images/lena.inr.gz -iy 128 -y 1 \
| cco -r > coupe.inr
cat | TEK_NOM=% cour | gv - <<EOF
1 coupe.inr 0
EOF
```

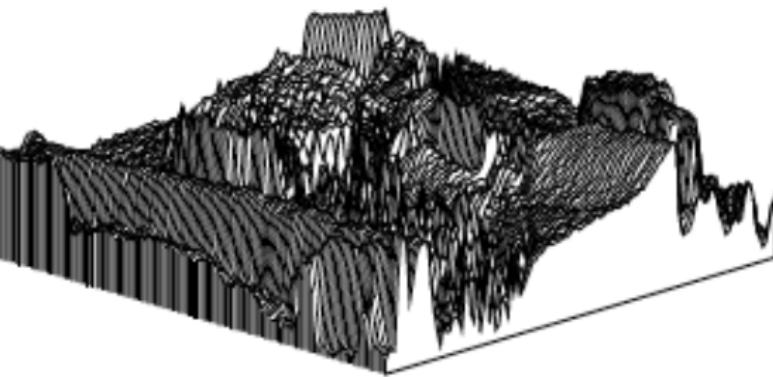


**FIGURE – Profil de la ligne
128**

Exemples

Exemple (Graphe de surface)

```
|| TEK_NOM=% p1 lena.inr \
   / gv -
```

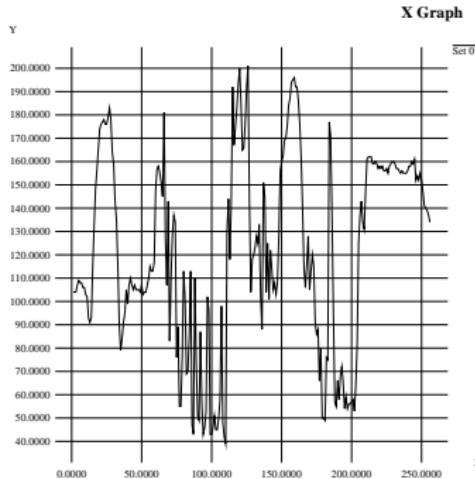


Exemples

- ▶ Ces commandes sont plutôt utilisées pour insertion dans des rapports imprimés. Pour la visualisation on peut préférer gnuplot, xgraph, ...

Exemple (Profil avec xgraph)

```
|| extg -iy 128 -y 1 lena.inr \
  | tpr -c -l 1 / nl / xgraph
```



Plan

Introduction

Le format Inrimage

Commandes Inrimage

Visualisation des images

Bibliothèque de développement

Tableau de format

Accès direct

Accès flottant

Compilation et exécution

Conclusion sur Inrimage

Deux modes d'accès aux images

- ▶ Après ouverture d'une image, il faut prévoir un tampon en mémoire pour stocker les valeurs numériques des pixels de cette image.
- ▶ Mais la taille de ce tampon dépend évidemment de la taille de codage de l'image.

Exemple

- ▶ *Une image codée sur 1 octet nécessite un tableau de caractères,*
- ▶ *une image codée sur 1 flottant nécessite un tableau de flottants,*
- ▶ *une image codée sur 1 bit nécessite un tableau de bit : ce qui n'existe pas en C d'ailleurs ! En pratique, on groupe par octet 8 pixels dans un tableau d'octets.*

Deux modes d'accès aux images (suite)

- ▶ Dès lors qu'on se restreint un codage cela ne pose pas de problème particulier.
- ▶ Dans cette UE, nous travaillerons essentiellement sur des images codées sur 1 octet. Ce mode d'accès est (quasi) direct : on transfère en mémoire des données tel qu'elles sont stockées dans le fichier image.
- ▶ Il peut être intéressant de s'affranchir des contraintes de taille de codage. Par exemple, cela permet d'utiliser un même algorithme pour une grande classe d'images.
- ▶ Inrimage propose un accès en mode "flottant" : quelquesoït le codage de l'image, les données sont transférées dans un tampon de flottant.

Tableau de format

- ▶ Pour décrire le format d'une image, Inrimage utilise un tableau de format à 9 éléments.
- ▶ Il est conventionnellement appelé `lfmt` et contient :

Elément	Macro	Description
<code>lfmt [I_DIMX]</code>	DIMX	nombre de valeurs par ligne
<code>lfmt [I_DIMY]</code>	DIMY	nombre de lignes dans l'image
<code>lfmt [I_BSIZE]</code>	BSIZE	nombre de bits ou d'octets pour coder une valeur
<code>lfmt [I_TYPE]</code>	TYPE	type de codage, peut valoir : REELLE, FIXE ou PACKEE
<code>lfmt [I_NDIMX]</code>	NDIMX	nombre de pixels par ligne
<code>lfmt [I_NDIMY]</code>	NDIMY	nombre de lignes par plan
<code>lfmt [I_NDIMV]</code>	NDIMV	nombre de valeurs par pixel
<code>lfmt [I_NDIMZ]</code>	NDIMZ	nombre de plans dans l'image
<code>lfmt [I_EXP]</code>	EXP	exposant de l'image

Tableau de format (suite)

- ▶ Les constantes `I_DIMX`, `I_DIMY`, ..., `I_EXP` valent respectivement 0, 1, ..., 9. Elles sont définies dans `inrimage/image.h`
- ▶ Les symboles `DIMX`, ..., `EXP` sont des macros définies dans `inrimage/image.h`.
- ▶ Par exemple `DIMX` est défini ainsi :

```
#define DIMX lfmt [I_DIMX]
```

Ces macros n'ont donc de sens que s'il existe un tableau nommé `lfmt` dans le code C !

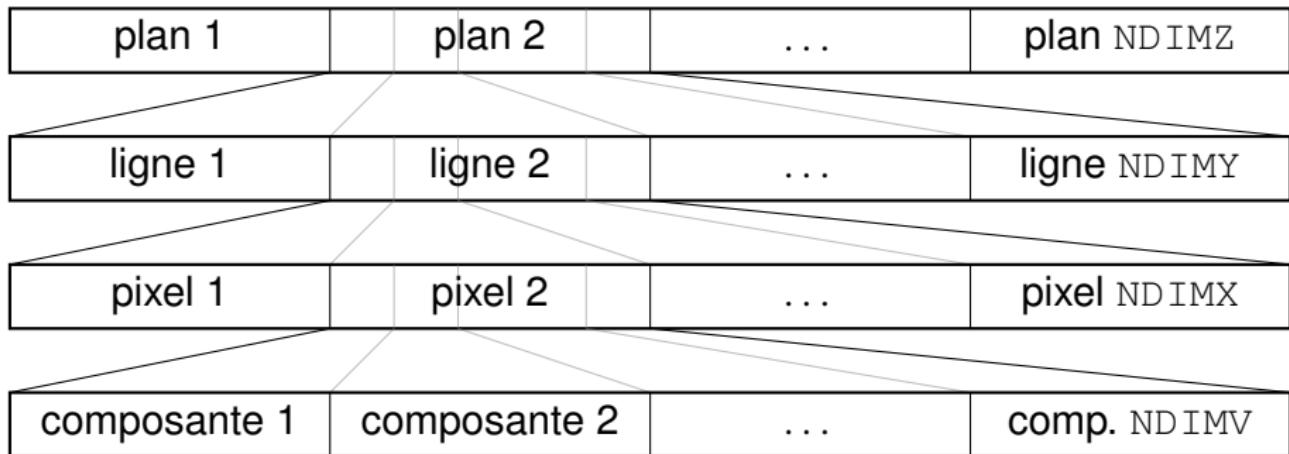
- ▶ Le type de ce tableau est `Fort_int`, il signifie Fortran Integer et est défini dans `inrimage/image.h`. Ce type permet à Inrimage de fonctionner avec Fortran, il est recommandé.

Tableau de format (suite)

- ▶ Les deux premiers éléments du tableau sont redondants car, on a toujours :
 1. $\text{DIMX} = \text{NDIMX} * \text{NDIMV}$
 2. $\text{DIMY} = \text{NDIMY} * \text{NDIMZ}$
- ▶ Ceci nous renseigne sur l'organisation structurelle des pixels dans une image Inrimage :
 - ▶ une image possède DIMY lignes en tout, ces lignes peuvent être regroupées en NDIMZ plans de NDIMY lignes,
 - ▶ une ligne est constituée de DIMX valeurs, regroupées en pixels contenant NDIMV valeurs, appelées également composantes.
 - ▶ Si $\text{NDIMV}=1$, l'image est dite scalaire, sinon elle est vectorielle. La valeur très courante est $\text{NDIMV}=3$ car elle est utilisée pour coder le format RGB.
 - ▶ Si $\text{NDIMZ}=1$, l'image est dite simple ou mono-plan, sinon on parle d'image multi-plan ou séquence d'images. $\text{NDIMZ}=3$ est parfois utilisé pour coder des images RGB !

Tableau de format (suite)

Structure des images en mémoire



- ▶ $NDIMV \times NDIMX \times NDIMY \times NDIMZ$ valeurs.

Tableau de format (suite)

- ▶ Les deux éléments `TYPE` et `BSIZE` permettent de déterminer le codage d'une valeur.
- ▶ Nous renconterons essentiellement deux types de codage :
 - ▶ **codage à virgule flottante** : correspond à `BSIZE=sizeof(float)` et `TYPE=REELLE` (constante définie dans `inrimage/image.h`);
 - ▶ **codage à virgule fixe** : correspond à `TYPE=FIXE` avec :
 - ▶ `BSIZE>0` pour indiquer une taille de codage en octet,
 - ▶ `BSIZE<0` pour indiquer une taille de codage en bit.
 - ▶ Il existe un troisième type d'image telle que `TYPE=PACKEE` : il s'agit d'une représentation compacte en bit. Nous n'utiliserons pas de telles images dans ce cours !

Tableau de format (suite)

Alignement des valeurs en mémoire

- ▶ Codage TYPE=FIXE sur n bits :
 - ▶ alignment sur multiple de 8 bits (=1 octet). Exemple pour un codage de 6 bits :

-	-	1	0	0	0	1	1	-	-	1	0	0	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

b7 b6 b5 b4 b3 b2 b1 b0 b7 b6 b5 b4 b3 b2 b1 b0

- ▶ il faut donc $\frac{n+7}{8}$ octets pour représenter n bits.
- ▶ Codage TYPE=PACKEE sur n bits :
 - ▶ les valeurs sont contiguës en mémoire :

1	0	0	0	1	1	1	0	0	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---

b5 b4 b3 b2 b1 b0 b5 b4 b3 b2 b1 b0

- ▶ on utilise donc exactement n bits pour coder une valeur ;
- ▶ attention : une ligne doit avoir une taille multiple de 8 bits ;
- ▶ il existe une fonction Inrimage qui permet de calculer la taille mémoire nécessaire pour stocker un nombre donné de lignes.

Tableau de format (suite)

- ▶ Codage à virgule fixe : le nombre est entier, mais on utilise un exposant pour en déduire une valeur flottante.
- ▶ `EXP` sert à la fois à représenter l'exposant lorsque le codage est à virgule fixe (`TYPE=FIXE` ou `TYPE=PACKEE`) et à indiquer si le codage est signé ou non.
- ▶ L'élément `EXP` sera ignoré dans ce cours (malgré son importance !)
- ▶ Inrimage possède encore d'autres paramètres pour décrire une image (voir la commande `par`). Comme ces paramètres sont rarement utilisés, il existe un format étendu (sous la forme d'une structure C) pour les décrire. Nous ne l'utiliserons pas dans ce cours.

Mode "direct"

```
/* exinrdir.c
 * LI316 - acces en mode direct
 */
#include <inrimage/image.h>

void process_image( unsigned char *buf, Fort_int Ifmt[], int param);

/* il est recommande de documenter sa commande */
char version[] = "1.0";
char commande[] = "[input][output]";
char detail[] = "Commande qui illustre une lecture et ecriture d'image";

int main( int argc, char **argv) {
    char filename[128]; // pour stocker le nom de l'image a traiter
    struct image *nf; // descripteur du fichier image
    Fort_int Ifmt[9]; // tableau de format
    unsigned char *buf; // tampon pour transferer les donnees
    int param = 0;

    // init Inrimage
    inr_init( argc, argv, version, commande, detail);
```

Mode "direct" (suite)

```
// lecture d'une option : elle servira à paramétrer l'algorithme
// de traitement de l'image (voir plus loin).
igetopt1( "-n", "%d", &param);

// lecture du nom d'image à traiter
infileopt( filename);

// ouverture de l'image, lecture puis fermeture

// ref man image : l'image est ouverte en mode 'existe'
nf = image_(filename, "e", "", lfmt);
// le tableau lfmt contient le format de l'image

if( BSIZE != 1 || TYPE != FIXE)
imerror( 22, "Codage non supporté\n");

// allocation pour stocker toute l'image
buf = i_malloc( DIMX*DIMY*sizeof(unsigned char));

// lecture de toute les lignes
c_lect( nf, DIMY, buf);

// fermeture
fermnf_( &nf);
```

Mode "direct" (suite)

```
// traitement de l'image (on suppose que buf est lu puis écrit)
process_image( buf, Ifmt, param);

// ouverture image de sortie, écriture puis fermeture

outfileopt( filename);
// ouverture en mode "creation" (ref man image), l'image de sortie
// aura le même format que l'image d'entrée.
nf = image_( filename, "c", "", Ifmt );
c_ecr( nf, DIMY, buf );
fermnf_( &nf );

i_Free( (void*)&buf );
return 0;
}
```

Mode direct : remarques

- ▶ Dans cet exemple, on s'est restreint à un codage particulier. S'il est facile de charger n'importe type d'image (il suffit de calculer la taille du tampon à partir des informations données dans `lfmt`), il est ensuite difficile de traiter l'image (fonction `process_image()`) car l'accès au pixel dans le tampon `buf` dépend du type !
- ▶ En C++, on masque cette difficulté en utilisant des TEMPLATES sur les fonctions MAIS, malgré tout, on a une grande variété de format de codage, pas seulement, `char`, `short` ou `float`. En particulier les codages au bit près sont difficiles à traiter.
- ▶ En pratique, en TME, on aura toujours des codages simples (1 ou 2 octets en virgule fixe, 4 octets en virgule flottante) et le type du tampon sera connu (`unsigned char`, `unsigned short` ou `float`).

Mode "flottant"

- ▶ Le principe est simple :
 - ▶ si l'image est codée en virgule flottante, l'image transfère directement les données dans le tampon de flottant.
 - ▶ si le codage est en virgule fixe, alors les valeurs sont normalisées en utilisant la taille de codage.

Exemple (codage 1 octet)

Une image codée sur un octet (8 bits) peut représenter $2^8 = 256$ valeurs. Il suffit donc de diviser chaque valeur par $2^8 - 1$ ainsi les valeurs varient entre 0 et 1 (1 correspond à 255).

- ▶ Pour les codages à virgule fixe, il y a alors 2 cas :
 - ▶ Image non signée : on divise par $2^b - 1$ où b est le nombre de bits pour coder une valeur. Les valeurs sont ramenées dans l'intervalle $[0, 1]$.
 - ▶ Image signée : les valeurs varient entre -2^{b-1} et $2^{b-1} - 1$. Donc on divise par $2^{b-1} - 1$ pour se ramener à des valeurs dans $[-1, 1]$.

Mode "flottant" (suite)

- ▶ On voit donc que lorsqu'on lit les images à virgule fixe en accès flottant, ce sont des réels ! Ils ont donc une mantisse et un exposant mais ce dernier est fixe et est calculé à partir de `lfmt[I_BSIZ]` et `lfmt[I_EXP]`.

Exemple (la commande `ical`)

*ical calcule les extrema d'une image ainsi que la valeur moyenne.
Si l'image est codée en virgule fixe, les valeurs calculées sont affichées en virgule flottante (car `ical` utilise un accès flottant).*

```
|| ical lena.inr 0.117647 0.499666 0.937255
|| par lena.inr
|| lena.inr -F=Inrimage -hdr=1 -x 256 -y 256 -f -o 1
```

Exercice

Quel est le niveau de gris maximal atteint dans cette image ?

Mode "flottant" (suite)

Réponse

Il faut multiplier le maximum calculé par `ical` et multiplier par la taille de codage - 1, soit :

```
|| echo '0.937255*(2^8-1)' | bc
|| 239.000025
```

Le maximum est 239.

Accès flottant : code C

```
/* exinrflt.c
 * LI316 - acces en mode flottant
 */
#include <inrimage/image.h>

void process_image_flt( float *buf, Fort_int lfmt[], float param);

char version[] = "1.0";
char commande[] = "[input][output]";
char details[] = "Commande qui illustre une lecture et une
                  écriture d'image en mode flottant";

int main( int argc, char **argv) {
    char filename[128];
    struct image *nf;
    Fort_int lfmt[9];
    float *buffer;           // le tampon est flottant
    float param = 0;

    inr_init( argc, argv, version, commande, details);
    igetopt1( "-n", "%f", &param);
    infileopt( filename);

    nf = image_( filename, "e", "", lfmt);
```

Accès flottant : code C (suite)

```
/* plus besoin de test de codage: tous les types d'image sont gérés */

buf = (float*)i_malloc( DIMX*DIMY*sizeof(float)) ;

/* lecture en mode flottant : fonction spécifique */
c_lecflt( nf, DIMY, buf) ;
fermnf_( &nf) ;

process_image_flt( buf, lfmt, param) ;

outfileopt( filename) ;
nf = image_ ( filename, "c", "", lfmt) ;
/* écriture en mode flottant : fonction spécifique */
c_ecrflt( nf, DIMY, buf) ;
fermnf_( &nf) ;

i_Free( (void*)&buf) ;
return 0;
}
```

Mise en œuvre des deux codes

- ▶ Les deux codes ouvrent une image, la traite et écrivent le résultat du traitement sur le disque. Le traitement n'est pas encore précisé : il sera l'objet des algorithmes que nous verrons à la suite de ce cours !
- ▶ Un algorithme nécessite souvent un ou plusieurs paramètres. Pour cela, Inrimage offre quelques fonctions pour gérer facilement la paramétrisation des commandes Inrimage (lire man igetopt).
- ▶ Un paramètre dans la ligne d'appel de la commande (dans le terminal texte) s'écrit de façon général :

-nom [valeur1] [valeur2]

le champ nom correspond au premier paramètre des commandes igetoptxxx().

Exemple

igetopt1 ("-n", "%f", ¶m); correspond au paramètre -n qui attend un paramètre de type flottant.

Mise en œuvre des deux codes (suite)

- ▶ Pour illustrer le fonctionnement des deux commandes, écrivons un algorithme très simple de traitement d'images : un seuillage.

```
/* pour le premier code, on a */
void process_image( unsigned char *buf, Fort_int lfmt[], int n) {
    long count = DIMX*DIMY;
    while( count --) *buf++ = (*buf > n)*255;
}

/* pour le second code, on a */
void process_image_flt( float *buf, Fort_int lfmt[], float n) {
    long count = DIMX*DIMY;
    while( count --) *buf++ = (*buf > n);
}
```

Les deux commandes sont compilées ainsi :

```
|| gcc exinrdir.c -o exinrdir -linrimage
|| gcc exinrflt.c -o exinrflt -linrimage -lm
```

- ▶ Si Inrimage n'est pas installé dans /usr/local, faire :

```
|| gcc exinrdir.c -o exinrdir `inrinfo --cflags --libs`
```

Mise en œuvre des deux codes (suite)

- ▶ Exécution des deux commandes :

```
./exinrdir -help
exinrdir-1.0, Inrimage Version 4.6.6
Usage: exinrdir [input] [output]
Commande qui illustre une lecture et écriture d'image
./exinrdir lena.inr lena1.inr -n 127
./exinrfilt lena.inr lena2.inr -n 0.5
```

- ▶ Visu et comparaison (commande so soustraie deux images) :

```
xvis -grp 2 lena?.inr
ical lena?.inr
lena1.inr:          0.000000      0.546585      1.000000
lena2.inr:          0.000000      0.546585      1.000000
```

Mise en œuvre des deux codes (suite)

```
|| xvis -nu -grp 2 -nhz 2 lena1.inr lena2.inr
```



Les deux images sont identiques :

```
|| so lena?.inr | ical
```

0.000000

0.000000

0.000000

Documentation d'Inrimage

- ▶ Documentation Html, point de départ :
 - ▶ ARI : /usr/local/share/doc/inrimage/index.html
 - ▶ WEB : <http://inrimage.gforge.inria.fr/WWW/index.html>
- ▶ Documentation dans le terminal :
 - ▶ Introduction aux commandes et au format Inrimage :
 - || **man** Inrimage
 - ▶ Introduction à la programmation Inrimage :
 - || **man** Intro
 - ▶ Retrouver le manuel d'une commande :
 - || **man** LISTE
 - ▶ Retrouver le manuel d'une fonction :
 - || **man** 3 LISTE

Plan

Introduction

Le format Inrimage

Commandes Inrimage

Visualisation des images

Bibliothèque de développement

Conclusion sur Inrimage

Conclusion sur Inrimage

- ▶ Logiciel de traitement numérique des images, spécialisé pour un fonctionnement en ligne de commandes et en script.
- ▶ Dans sa catégorie, il est quasiment le seul.
- ▶ Simple, efficace, bas-niveau, portable. Permet de traiter quasiment tout type d'image existant effectivement sur le marché.
- ▶ La bibliothèque d'accès aux images est également simple à mettre en œuvre.
- ▶ L'utilisation basique d'Inrimage nécessite donc peu d'apprentissage (à la différence de `matlab` par exemple).

Annexes

Plan

Configuration d'Inrimage

Classement thématique des commandes

Configuration d'Inrimage

- ▶ Le comportement par défaut d'Inrimage est réglé par plusieurs fichiers de configuration.
- ▶ La commande `inrinfo` donne des informations à ce sujet :

```
|| inrinfo
```

```
|| Inrimage Version 4.6.3
```

```
|| INR_HOME /usr/share/inrimage
```

```
|| INR_EXTFILE /usr/share/inrimage/etc/inr_extfile
```

- ▶ La ligne `INR_HOME` indique où est installé Inrimage. Ce répertoire contient le répertoire `etc` :

```
|| ls /usr/share/inrimage/etc
```

```
|| inr_extfile Inrimage4.conf xvis.sh
```

- ▶ le fichier `inr_extfile` contient les règles de conversion lorsqu'une commande lit une image non Inrimage.
- ▶ le fichier `Inrimage4.conf` contient des options globales (par exemple : taille de l'entête, nombre max de lignes d'historique, ...).

Configuration d'Inrimage (suite)

- ▶ La ligne `INR_EXTFILE` indique quel fichier de filtre est utilisé.
- ▶ En effet, il peut y en avoir plusieurs :
 - ▶ Celui défini dans `INR_HOME/etc`.
 - ▶ Celui défini dans `~/.inrimage`. S'il existe, ce dernier prend le pas sur le premier.
- ▶ Enfin les deux variables `INR_HOME` et `INR_EXTFILE` sont définies par `Inrimage.conf` dans le répertoire d'installation d'Inrimage ou bien par des variables d'environnement.

Exemple (Des configurations multiples)

```
INR_HOME=/ inrinfo
INR_HOME /
INR_EXTFILE unknown
INR_HOME=~/ .bashrc inrinfo
INR_HOME /usr/share/inrimage
INR_EXTFILE /home/bereziat/.bashrc
mkdir -p ~/.inrimage; touch ~/.inrimage/inr_extfile
inrinfo
INR_HOME /usr/share/inrimage
INR_EXTFILE /home/bereziat/.inrimage/inr_extfile
```

Configuration d'Inrimage

Fichier de conversion

- Le fichier `inr_extfile` indique à Inrimage comment lancer les programmes de conversion :

```
inrinfo -ext
/usr/local/share/inrimage/etc/inr_extfile
cat /usr/local/share/inrimage/etc/inr_extfile
.gif,.GIF ::gif2inr
.rle :: rle2inr
.bmp,.pic :: bmp2inr
.xbm :: xbm2inr
.hdr :: hdr2inr
.img :: img2inr
.vif,.xv:0 ab 1: viff2inr
.xwd :: d2im -hdr=8 -k 1000
.ras :: ras2inr
.tif,.tiff:0 4d 4d,0 49 49:tiff2inr -d2z:/usr/local/bin/tiff2inr
.png: :png2inr -noalpha:/usr/local/bin/png2inr
.pbm,.pgm,.ppm,.pnm,.pam :: pam2inr
.jpg, .JPG, .jpeg:: jpg2inr
```

- Si on n'est pas administrateur du système, pensez à créer son propre fichier placé dans `~/.inrimage/inr_extfile`.

Liste des variables d'environnement

INR_HOME	Le répertoire d'installation de Inrimage (/usr/local/inrimage par défaut).
IMAGE_OUT	Type d'en-tête (voir option -F=).
IMAGE_HDRL	Taille d'en-tête (voir option -hdr=).
IMAGE_HIST	Nombre maximum d'événements du mécanisme d'historique.
TEK_NOM	Nom de fichier pour les sorties graphiques. Doit être définie comme TEK_NOM=% pour une sortie postscript dans stdout.
INR_EXTFILE	Adresse du fichier de définition des conversions pour les fichiers non Inrimage.
INR_KBMEM	Taille en Kilo-octets de la mémoire utilisée dans la plupart des commandes Inrimage pour les tableaux. Cette valeur n'est utilisée qu'en l'absence de l'option -k.

Exemple de commande interactive

- ▶ La commande **vect** permet de transformer une image vectorielle en image multiplan ou en plusieurs images (et vis versa).

```
par lena.inr
lena.inr -F=Inrimage -hdr=1 -x 256 -y 256 -v 3 -f -o 1
vect
NB IMAGES D'ENTREE (0 EQ 1) 0_
IMAGE D'ENTREE
fichier lena.inr_
SORTIE SUR FICHIERS SEPARES? (O/N) o_
IMAGE DE SORTIE
IMAGE 1
fichier r_
```

Commande interactive

suite

```
IMAGE 2
fichier v_
IMAGE 3
fichier b_
par r v b
r           -F=Inrimage   -hdr=1 -x 256   -y 256   -f   -o 1
v           -F=Inrimage   -hdr=1 -x 256   -y 256   -f   -o 1
b           -F=Inrimage   -hdr=1 -x 256   -y 256   -f   -o 1
```

- ▶ Une commande interactive lit dans l'entrée standard les ordres :

```
|| echo 0 lena.inr o r g b | vect
```

Plan

Configuration d'Inrimage

Classement thématique des commandes

Manipulation de format

Conversions d'image

Création, manipulation d'image

Opérations arithmétiques et logiques

Filtrage d'image

Représentation graphique et impression d'image

Traitement des contours

Autres commandes

Classement thématique des commandes

- ▶ Les commandes sont classés thématiquement dans la documentation :

|| man LISTE

- ▶ Les catégories :

1. Manipulation de format (page 92)
2. Conversion d'image (page 93)
3. Création, manipulation d'image (page 95)
4. Opérations arithmétiques et logiques (page 97)
5. Filtrage d'image (page 99)
6. Représentation graphique et impression d'image (page 100)
7. Traitement des contours (page 101)
8. Autres commandes (page 102)

Manipulation de format

Commande	Manuel	Description
cco	cco(1i)	passage d'un codage d'image à un autre
ical	ical(1i)	calcul des extrema d'une image
ifmt	ifmt(1i)	modification de l'en-tête d'images
cnvcol	cnvcol(1i)	conversion d'images niveaux de gris ou en couleur
par	par(1i)	impression sur stdout des paramètres de format d'images
cpar	cpar(1i)	print informations on the image header in a C-style
itest	itest(1i)	perform some simple tests on image headers.

Conversions d'image

Commande	Manuel	Description
inr2bmp	inr2bmp(1i)	Inrimage vers BMP.
inr2gif	inr2gif(1i)	Inrimage vers GIF.
inr2jp2	inr2jp2(1i)	Inrimage vers JPEG 2000.
inr2jpg	inr2jpg(1i)	Inrimage vers JPEG.
inr2mpg	inr2mpg(1i)	Construction d'animations mpeg.
inr2pam	inr2pam(1i)	Inrimage vers NetPbm.
inr2png	inr2png(1i)	Inrimage vers PNG.
inr2ppm	inr2ppm(1i)	Inrimage vers PPM/PGM.
inr2tiff	inr2tiff(1i)	Inrimage vers TIFF.
inr2viff	inr2viff(1i)	Inrimage vers VIFF (Khoros).
inr2xbm	inr2xbm(1i)	Inrimage en bitmap X.
bmp2inr	bmp2inr(1i)	BMP vers Inrimage.
gif2inr	gif2inr(1i)	GIF vers Inrimage.

Conversions d'image (suite)

Commande	Manuel	Description
jp22inr	jp22inr(1i)	JPEG 2000 vers Inrimage.
jpg2inr	jpg2inr(1i)	JPEG vers Inrimage.
mpg2inr	mpg2inr(1i)	conversion MPEG vers Inrimage.
png2inr	png2inr(1i)	PNG vers Inrimage.
ppm2inr	ppm2inr(1i)	PPM/PGM vers Inrimage.
ps2inr	ps2inr(1i)	Récupération d'une image dans un fichier PostScript.
ras2inr	ras2inr(1i)	sunraster vers Inrimage.
tiff2inr	tiff2inr(1i)	TIFF vers Inrimage.
viff2inr	viff2inr(1i)	VIFF (Khoros) vers Inrimage.

Création, manipulation d'image

Commande	Manuel	Description
cim	cim(1i)	entrée de valeurs ascii dans une image.
create	create(1i)	création d'en-têtes d'images
ge	ge(1i)	génération d'images monodimensionnelle partir de diverses fonctions.
eg	eg(1i)	égalisation de l'histogramme d'une image
ext	ext(1i)	obsolète : voir extg
extg	extg(1i)	extraction d'une sous-image
his	his(1i)	calcul de l'histogramme d'une image.
icompose	icompose(1i)	composition d'image à partir de plusieurs
ilsfonts	ilsfonts(1i)	Liste les polices de caractères utilisables
inrcat	inrcat(1i)	copie d'images avec suppression d'en-tête
itext	itext(1i)	écriture de textes dans une image.

Création, manipulation d'image (suite)

Commande	Manuel	Description
fzoom	izoom(1i)	agrandissement ou réduction d'une image réel
izoom	izoom(1i)	agrandissement ou réduction d'une image 8 b
mel	mel(1i)	mélange de 2 images.
melg	melg(1i)	mélange de 2 images avec test sur les valeur
raz	raz(1i)	mise a zéro d'une image.
rot	rot(1i)	rotation d'une image autour d'un axe // Z et d'un vecteur ligne.
se	se(1i)	calcul d'une image par produit d'un vecteur co
to	to(1i)	construction d'une image par rotation d'une li
tran	tran(1i)	symétries et rotations sur une image
vect	vect(1i)	passage d'une image vectorielle ou multiplan plusieurs images et vice versa.

Opérations arithmétiques et logiques

Commande	Manuel	Description
bi	arit1(1i)	addition d'un scalaire à 1 image
car	arit1(1i)	carré d'une image.
carflo	carflo(1i)	conversion de paramètres entiers en flottant.
ad	arit2(1i)	addition point à point de 2 images.
di	arit2(1i)	division point à point de 2 images.
exp	arit1(1i)	exponentielle d'une image.
lo	arit1(1i)	logarithme d'une image.
logic	logic(1i)	opérations de logique binaire sur une image.
max	arit2(1i)	maximum point à point de 2 images.
mb	arit1(1i)	masque bas d'une image.
mh	arit1(1i)	masque haut d'une image.
min	arit2(1i)	minimum point à point de 2 images.

Opérations arithmétiques et logiques (suite)

Commande	Manuel	Description
mo	arit1(1i)	module d'une image.
mu	arit2(1i)	multiplication point à point de 2 images.
muls	muls(1i)	seuillage d'une image par seuils multiples.
norma	norma(1i)	normalisation d'une image.
ra	arit1(1i)	racine carrée d'une image.
sba	arit1(1i)	seuil bas d'une image.
sc	arit1(1i)	multiplication d'une image par un scalaire.
sd	arit1(1i)	division d'une image par un scalaire.
sha	arit1(1i)	seuil haut d'une image.
so	arit2(1i)	soustraction point à point de 2 images.
vb	arit1(1i)	seuil bas variable d'une image.
vh	arit1(1i)	seuil haut variable d'une image.

Filtrage d'image

Commande	Manuel	Description
ce	ce(1i)	recentrage d'une image.
conv	conv(1i)	convolution d'une image par un noyau de taille
detc	detc(1i)	détection de contours dans une image.
ddf	rdf(1i)	FFT directe sur une image complexe.
dmo	rmo(1i)	FFT monodimensionnelle complexe directe.
fmoy	fmoy(1i)	convolution d'une image par un filtre moyen.
idf	rdf(1i)	FFT inverse sur une image complexe.
imo	rmo(1i)	FFT monodimensionnelle complexe inverse.
ma	ri(1i)	passage de (module, phase) à (partie réelle, im
med	med(1i)	filtre médian.
rdf	rdf(1i)	FFT directe sur une image réelle.
ri	ri(1i)	passage de (partie réelle, imaginaire) à (modu
rmo	rmo(1i)	FFT monodimensionnelle réelle directe.

Représentation graphique et impression d'image

Commande	Manuel	Description
cou	cou(1i)	représentation de coupes dans une image, sous format graphique Tektronix ou PostScript.
cour	cour(1i)	représentation de plusieurs courbes sur un même dessin, sous format graphique Tektronix ou PostScript.
d2im	d2im(1i)	conversion de dump d'écran X11 en image IFF.
im2ps	im2ps(1i)	conversion d'images monochromes en PostScript.
im2psc	im2psc(1i)	conversion d'images couleur en PostScript.
pl	pl(1i)	représentation graphique d'une image sous format PostScript.
tek2ps	tek2ps(1i)	conversion de graphiques Tektronix en PostScript.
tpr	tpr(1i)	impression des valeurs des pixels d'une image.
gvis	gvis(1i)	visualisation d'images sous GTK.
xvis	xvis(1i)	visualisation d'images sous X-window (version 1.0).

Traitement des contours

Commande	Manuel	Description
aff	aff(1i)	affinage d'une image.
anac	anac(1i)	suivi de contours dans une image seuillée.
apol	apol(1i)	approximation polygonale de contours.
ero	ero(1i)	érosion ou dilatation d'image.
fillc	fillc(1i)	remplissage de contours dans une image.
visc	visc(1i)	visualisation d'un fichier de contours (format Tektronix ou PostScript).

Autres commandes

Commande	Manuel	Description
carflo	carflo(1)	transcodage de valeurs codées sur 1 octet en flottant
fixflo	fixflo(1)	transcodage de valeurs à virgule fixe en valeurs à virgule flottante
flofix	flofix(1)	transcodage de valeurs à virgule flottante en valeurs à virgule fixe
inrinfo	inrinfo(1)	infos sur l'installation Inrimage courante
inrfulltest	inrfulltest(1)	test pousse de l'installation Inrimage