HMIN318M – TP 0 (3 heures)

Gérard Subsol – 14 septembre 2017

Ce TP vous fera découvrir les mécanismes de base (lecture-écriture, stockage, filtrage) du traitement d'images 3D ainsi que des outils de visualisation 3D (MPR, interaction avec la souris).

Ce TP ne sera pas évalué mais il servira de base pour les TPs et les projets ultérieurs.
Néanmoins, la participation active pendant le TP sera prise en compte dans l'évaluation finale.
Ce TP peut être effectué en binôme.

Etudiants Master Infomatique

Découverte de Cimg (http://cimg.eu/ **)**
Manuel de référence disponible à http://cimg.eu/CImg_reference.pdf

CImg est une bibliothèque C++, basée sur l'utilisation (basique) de templates. Elle définit un nombre minimal de classes (4 au total) et de fonctions permettant une manipulation aisée d'images dans un programme C++, en gérant par exemple les entrée-sorties, l'affichage/l'interaction, le filtrage, la manipulation géométrique, le dessin de primitives, etc.... C'est une bibliothèque libre et multi-plateforme, développée dans l'équipe Image du laboratoire de recherche (CNRS) GREYC, à Caen/France. Son développement a commencé fin 1999, à l'INRIA de Sophia-Antipolis.

CImg a été conçue avant tout dans un but de simplicité d'utilisation. Elle est adaptée lorsque l'on cherche, par exemple, à faire du prototypage d'algorithmes de traitement d'image. Elle est relativement légère, contenue principalement dans un fichier d'en-tête C++ CImg.h à inclure en début de source.

CImg est générique. Ses classes permettent de manipuler aussi bien des signaux 1D que des images 2D ou 3D multi-valuées (couleurs par exemple, ou avec un nombre quelconque de composantes), ainsi que des séquences d'images (séquences temporelles typiquement). Les valeurs des pixels des images sont des types templates, et il est donc possible de gérer de manière transparente des images à 8 bits ou 16 bits par composante, mais aussi à valeurs flottantes.

Un programme Cimg se présente comme :

```
#include "CImg.h"
using namespace cimg_library;
….
main
{
}
```

Et se compile comme suit :

- Microsoft Visual C++ 6.0, Visual Studio.NET and Visual Express Edition : Use project files and solution files provided in the CImg Library package (directory 'compilation/') to see how it works.
- g++ (MingW windows version) : g++ -o hello_word.exe hello_word.cpp -O2 -lgdi32
- g++ (Linux version) : g++ -o hello_word.exe hello_world.cpp -O2 -L/usr/X11R6/lib -lm -lpthread -lX11
- g++ (Mac OS X version) : g++ -o hello_word.exe hello_world.cpp -O2 -lm -lpthread -L/usr/X11R6/lib -lm -lpthread -lX11
- Dev-Cpp : Use the project file provided in the CImg library package to see how it works.

1. Lecture et stockage d'une image 3D

- Ecrire un programme qui va lire les images au format Analyze.

  Une *image au format Analyze est constituée d'un en-tête (.hdr) qui contient le format des valeurs des voxels, les dimensions de l'image ainsi que la taille du voxel et de l'image (.img) qui est codée en bitmap, c'est-à-dire avec les valeurs des voxels en binaire (sur 1 à 4 octets suivant leur format) qui sont listées les unes derrière les autres. . Il est donc indispensable que les deux fichiers soient dans le même répertoire.*

- Afficher la taille de l'image (dimension X, Y, Z), la dimension du voxel et la valeur minimale et maximale.

- Afficher la valeur du voxel (256,256,12) qui doit être 986 pour l'image knix.hdr.

  Voir l'annexe 1 qui est extraite de la documentation en ligne de Cimg.

2. Visualisation d'une image 3D

- Ecrire un programme qui affiche à l'écran l'image test en coupes axiales.
- Ecrire un programme qui affiche à l'écran l'image test en MPR (en haut à gauche : coupes originales axiales xy, en haut à droite coupes zy et en bas à gauche xz).
- Dans les deux cas, on utilisera la molette de la souris pour faire défiler les coupes et on pourra utiliser la touche 'esc' pour quitter.

  Voir l'annexe 2 qui est extraite de la documentation en ligne de Cimg.

3. Filtrage simple

- Rajouter une fonctionnalité de filtrage simple (par exemple *blur*) quand on appuie sur la touche 'M'.

  Voir l'annexe 2 qui est extraite de la documentation en ligne de Cimg.

4. Interface avancée

- Améliorer l'interface en :
  - Récupérant la position de la souris
  - Calculant dans l'image MPR la position du curseur (c'est-à-dire les coordonnées dans l'image 3D)
  - Ajoutant du texte pour afficher la position du curseur
  - Prenant en compte la position de la souris pour faire défiler les coupes dans le bon quadrant du MPR
  - Utilisant le clic milieu pour pointer dans un quadrant ce qui modifie les 2 autres quadrants…

5. Visualisation volumique

- Programmer une visualisation volumique (MIP, minIP, AIP) suivant un des axes (x,y,z) avec un affichage dans une nouvelle fenêtre.

Annexe 1

How pixel data are stored with CImg (*http://cimg.eu/reference/group__cimg__storage.html*)

First, CImg<T> are *very* basic structures, which means that there are no memory tricks, weird memory alignments or disk caches used to store pixel data of images. When an image is instanced, all its pixel values are stored in memory at the same time (yes, you should avoid working with huge images when dealing with CImg, if you have only 64kb of RAM).

A CImg<T> is basically a 4th-dimensional array (width,height,depth,dim), and its pixel data are stored linearly in a single memory buffer of general size (width*height*depth*dim). Nothing more, nothing less. The address of this memory buffer can be retrieved by the function CImg<T>::data(). As each image value is stored as a type T (T being known by the programmer of course), this pointer is a 'T*', or a 'const T*' if your image is 'const'. so, 'T *ptr = img.data()' gives you the pointer to the first value of the image 'img'. The overall size of the used memory for one instance image (in bytes) is then 'width*height*depth*dim*sizeof(T)'.

Now, the ordering of the pixel values in this buffer follows these rules : The values are *not* interleaved, and are ordered first along the X,Y,Z and V axis respectively (corresponding to the width,height,depth,dim dimensions), starting from the upper-left pixel to the bottom-right pixel of the instane image, with a classical scanline run.

Detailed Description (*http://cimg.eu/reference/structcimg__library_1_1CImg.html*)

template<typename T>
struct cimg_library::CImg< T >

Class representing an image (up to 4 dimensions wide), each pixel being of type T.

This is the main class of the CImg Library. It declares and constructs an image, allows access to its pixel values, and is able to perform various image operations.

Image representation

A CImg image is defined as an instance of the container CImg<T>, which contains a regular grid of pixels, each pixel value being of type T. The image grid can have up to 4 dimensions: width, height, depth and number of channels. Usually, the three first dimensions are used to describe spatial coordinates (x,y,z), while the number of channels is rather used as a vector-valued dimension (it may describe the R,G,B color channels for instance). If you need a fifth dimension, you can use image lists CImgList<T> rather than simple images CImg<T>.

Thus, the CImg<T> class is able to represent volumetric images of vector-valued pixels, as well as images with less dimensions (1d scalar signal, 2d color images...). Most member functions of the class CImg<T> are designed to handle this maximum case of (3+1) dimensions.

Concerning the pixel value type T : fully supported template types are the basic C++ types : unsigned char, char, short, unsigned int, int, unsigned long, long, float, double, ... Typically, fast image display can be done using CImg<unsigned char> images, while complex image processing algorithms may be rather coded using CImg<float> or CImg<double> images that have floating-point pixel values. The default value for the template T is float. Using your own template types may be possible. However, you will certainly have to define the complete set of arithmetic and logical operators for your class.

Image structure

The CImg<T> structure contains six fields:

- _width defines the number of columns of the image (size along the X-axis).

- _height defines the number of rows of the image (size along the Y-axis).
- _depth defines the number of slices of the image (size along the Z-axis).
- _spectrum defines the number of channels of the image (size along the C-axis).
- _data defines a pointer to the pixel data (of type T).
- _is_shared is a boolean that tells if the memory buffer data is shared with another image.

You can access these fields publicly although it is recommended to use the dedicated functions width(), height(), depth(), spectrum() and ptr() to do so. Image dimensions are not limited to a specific range (as long as you got enough available memory). A value of 1 usually means that the corresponding dimension is flat. If one of the dimensions is 0, or if the data pointer is null, the image is considered as empty. Empty images should not contain any pixel data and thus, will not be processed by CImg member functions (a CImgInstanceException will be thrown instead). Pixel data are stored in memory, in a non interlaced mode (See How pixel data are stored with CImg.).

Image declaration and construction

Declaring an image can be done by using one of the several available constructors. Here is a list of the most used:

Construct images from arbitrary dimensions:

- CImg<char> img; declares an empty image.
- CImg<unsigned char> img(128,128); declares a 128x128 greyscale image with unsigned char pixel values.
- CImg<double> img(3,3); declares a 3x3 matrix with double coefficients.
- CImg<unsigned char> img(256,256,1,3); declares a 256x256x1x3 (color) image (colors are stored as an image with three channels).
- CImg<double> img(128,128,128); declares a 128x128x128 volumetric and greyscale image (with double pixel values).
- CImg<> img(128,128,128,3); declares a 128x128x128 volumetric color image (with float pixels, which is the default value of the template parameter T).

Note : images pixels are not automatically initialized to 0. You may use the function fill() to do it, or use the specific constructor taking 5 parameters like this : CImg<> img(128,128,128,3,0); declares a 128x128x128 volumetric color image with all pixel values to 0.

Loading Analyze images

```
CImg<T>& load_analyze   ( const char *const  filename,
                          float *const       voxel_size = 0
                          )
```

Load image from an ANALYZE7.5/NIFTI file.

Parameters:

	filename	Filename, as a C-string.
[out]	voxel_size	Pointer to the three voxel sizes read from the file.

Minimum and maximum

T & min ()

Return a reference to the minimum pixel value.

T & max ()

Return a reference to the maximum pixel value.

Annexe 2

CImgDisplay Struct Reference Detailed Description
(*http://cimg.eu/reference/structcimg__library_1_1CImgDisplay.html* **)**

Allow to create windows, display images on them and manage user events (keyboard, mouse and windows events).

CImgDisplay methods rely on a low-level graphic library to perform: it can be either X-Window (X11, for Unix-based systems) or GDI32 (for Windows-based systems). If both libraries are missing, CImgDisplay will not be able to display images on screen, and will enter a minimal mode where warning messages will be outputed each time the program is trying to call one of the CImgDisplay method.

Example (***http://cimg.eu/reference/group__cimg__tutorial.html*** **)**

```
#include "CImg.h"
  using namespace cimg_library;

  int main() {
    CImg<unsigned char> image("lena.jpg"), visu(500,400,1,3,0);
    const unsigned char red[] = { 255,0,0 }, green[] = { 0,255,0 }, blue[] = { 0,0,255 };
    image.blur(2.5);
    CImgDisplay main_disp(image,"Click a point"), draw_disp(visu,"Intensity profile");
    while (!main_disp.is_closed() && !draw_disp.is_closed()) {
      main_disp.wait();
      if (main_disp.button() && main_disp.mouse_y()>=0) {
        const int y = main_disp.mouse_y();
        visu.fill(0).draw_graph(image.get_crop(0,y,0,0,image.width()-1,y,0,0),red,1,1,0,255,0);
        visu.draw_graph(image.get_crop(0,y,0,1,image.width()-1,y,0,1),green,1,1,0,255,0);
        visu.draw_graph(image.get_crop(0,y,0,2,image.width()-
1,y,0,2),blue,1,1,0,255,0).display(draw_disp);
      }
    }
    return 0;
  }
```

To display a 3D image

CImg< T > get_slice (const int z0) const
Return specified image slice.

CImg< T > get_projections2d (const unsigned int x0, const unsigned int y0, const unsigned int z0) const
Generate a 2d representation of a 3d image, with XY,XZ and YZ views.

To blur a 3D image

Blur image.
Parameters
sigma_x Standard deviation of the blur, along the X-axis.
sigma_y Standard deviation of the blur, along the Y-axis.
sigma_z Standard deviation of the blur, along the Z-axis.
boundary_conditions Boundary conditions. Can be { false=dirichlet | true=neumann }.