Informations sur la bibliothèque CImg

CImg est une bibliothèque open source de traitement d'image en C++. Elle se présente sous la forme d'un fichier entête unique 'CImg.h' contenant un ensemble minimal de classes et de méthodes pour la gestion de fichier image, le traitement et l'affichage des images. Cette bibliothèque est portable (Windows, Unix, MacOS,...), légère et facile à utiliser.

Nous allons utiliser cette bibliothèque dans les mini-projets pour pouvoir gérer une petite interface graphique permettant de dessiner des éléments ou d'afficher des images.

1 Installation de CImg

Il suffit simplement de copier le fichier 'CImg.h' dans le dossier de votre projet C++ et de l'inclure dans vos fichiers sources sous la forme :

```
#include "CImg.h"
```

Les classes et les fonctions de la bibliothèque peuvent ensuite être utilisées.

Pour vérifier que la bibliothèque fonctionne sous votre environnement de développement, vous pouvez :

1. Récupérer l'archive `CImg.zip` sur le portail.
2. Décompresser cette archive dans un dossier de travail.
3. Compiler un des exemples du dossier de travail.

2 Utilisation de la bibliothèque

La documentation des classes et des fonctions est disponible sur le site web de CImg, en cliquant ici.

Plusieurs exemples simples d'utilisation en lien avec les projets sont fournis dans l'archive. Ils sont repris ci-dessous.

Sous Windows, pour pouvoir utiliser des formats de fichiers JPG, GIF, PNG, TIF,... il faut installer la bibliothèque ImageMagick (voir onglet *download* sur le site internet d'ImageMagick.)

2.1 Chargement et affichage d'une image

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

int main() {

// Declare an image
  CImg<unsigned char> image;
// Load an image file into it
  image.load("lena.ppm");

// Declare a display and dispay the image into it
  CImgDisplay disp(image,"Image");

// Wait for the display window to close
  while (!disp.is_closed()) {
    disp.wait();
    }

  return 0;
}
```

2.2 Utilisation d'une fonction de filtrage

```
/*
example2.cpp

Using the function convolve
*/
#include <iostream>
#include "CImg.h"

using namespace std;
using namespace cimg_library;

int main() {

// Declare a float image
  CImg<float> image;
// Load an image file into it
  image.load("lena.ppm");
  cout<<"Image loaded..."<<endl;

// Define a 5x5 convolution mask
  CImg<float> mask(5,5,1,1,1.);
```

```
// Print values of the mask
  cout<<"Values of the mask image : ["<<endl;
  for (int y=0;y<mask.height();y++) {
    cout<<"  ";
    for (int x=0;x<mask.width();x++) cout<<mask(x,y)<<" ";
    cout<<endl;
    }
  cout<<"]"<<endl;

// Convolve and normalize the image
  CImg<float> res = image.get_convolve(mask).normalize(0,255);

// Create an image list for side by side display
  CImgList<float> list(image,res);

// Declare a display and dispay the two images
  CImgDisplay disp(list,"Convolution by a 5x5 mean kernel");
  cout<<"<< Close the window to end the program >>"<<endl;

// Wait for the display window to close
  while (!disp.is_closed()) {
    disp.wait();
    }

  return 0;
}
```

2.3 Utilisation du graphique pour gérer l'affichage d'un plateau de jeu

```
/*
example3.cpp

Using drawing functions and mouse input
*/
#include <iostream>
#include "CImg.h"

using namespace std;
using namespace cimg_library;

int main() {

// Usefull colors
unsigned char
  grid_color[3]={0,0,255},
  play1_color[3]={204,0,0},
```

```
  play2_color[3]={0,102,0};

// Declare an image to draw the grid
  CImg<unsigned char> grid(300,300,1,3,255);
  for (int i=1;i<3;i++) grid.draw_line(0,100*i,300,100*i,grid_color);
  for (int j=1;j<3;j++) grid.draw_line(100*j,0,100*j,300,grid_color);

// Declare a display to draw the scene
  CImgDisplay disp(grid,"Tic-tac-toe",0,false,false);

// Center the window on the screen
   disp.move((CImgDisplay::screen_width() - disp.width())/2,
             (CImgDisplay::screen_height() - disp.height())/2);

// Declare an image to display the scene
CImg<unsigned char> scene=grid;
// Usefull variables
int player=1;
int table[3][3]={0,0,0,0,0,0,0,0,0};

// Main loop, exit if the display window is closed or if ESC or Q key is hit
  while (!disp.is_closed() && !disp.is_keyESC() && !disp.is_keyQ()) {

    // Display the scene
    scene.display(disp);

    // Detect mouse click
    if (disp.button()&1) { // Left button clicked.
      int i=3*disp.mouse_y()/disp.height();
      int j=3*disp.mouse_x()/disp.width();

      if (table[i][j]==0) {
        // Player 1, draw circle
        if (player==1) {
          scene.draw_circle(50+j*100,50+i*100,40,play1_color,1,~0U);
          table[i][j]=1;
        }
        // Player 2, draw cross
        else {
          scene.draw_line(10+j*100,10+i*100,90+j*100,90+i*100,play2_color);
          scene.draw_line(10+j*100,90+i*100,90+j*100,10+i*100,play2_color);
          table[i][j]=2;
        }
      player=3-player;
      }
      // If the scene is full, reset the game
      else {
        bool find_void=false;
```

```
        int *t=&table[0][0];
        for (int k=0; !find_void && k<9;k++) if (t[k]==0) find_void=true;
        if (!find_void) {
          for(int k=0;k<9;k++) t[k]=0;
          scene=grid;
        }
      }
    }
    disp.wait();

    // Handle window resize
    if (disp.is_resized()) disp.resize();
    }

  return 0;
}
```