Image Analysis and Computer Vision
Short Tutorial to the CImg & Argstream Libraries

March 22, 2007

1 The CImg Library

The CImg library is a modern image library for c++ which greatly simplifies programming, as one can access the image pixels naturally by their 2d position, i.e. imgVariable(x,y), contrary to the somewhat tedious linear accessing, i.e. imgVariable[x+y*width], of low level libraries.

1.1 Declaration of Images

First of all, to use the libarary in your code you have to include one single header file "CImg.h" and indicate that its namespace "cimg_library" has to be used. The declaration of an image is then simply given by *CImg<type>* where the type stands for the variable type used for the pixels. We suggest to use the type *float*. Instead of only declaring an image, one can directly allocate the desired image size, initialize the image from a file or clone it from a previous image.

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

CImg<float> myEmptyFloatImg        /* pure declaration of image (no size) */
CImg<float> myFloatImg (300,200);      /* 300x200 float pixel image */
CImg<float> myFloatImg1 ("demo.png");  /* directly loading a file */
CImg<float> myFloatImg2 (myFloatImg);  /* copying a previous image */
```

1.2 Loading of Images

Loading and saving of images is very simple and a vast of image formats are supported (basically everything the command 'convert' can handle):

```
string testFilename="testImage.png";
myFloatImg.load (testFilename.c_str());    /* loading from file */
myFloatImg.save ("outputImage.png");  /* saving to file */
```

Obviously, when loading an image any content in the variable myFloatImg will be lost. Moreover, as this library is ready to deal with color images – but in this course we mainly deal with greyscale images – you might need to convert an image to grey scale which is achieved by

```
/* conversion to greyscale images (does not change non-color images) */
myFloatImg = myFloatImg.get_norm_pointwise(1) / (float)myFloatImg.dimv();
```

As it does not change an image which is already in greyscale format, we suggest that you always do that unless you really want to deal with colour images.

1.3 Image inspection

You might also be interested in viewing the content of an image variable during the execution of your program which is achieved by

```
myFloatImg.display ("Window Title");  /* displaying the image */
```

which opens a window (with specified title) on your screen. The program execution is then blocked until you press any key (while that window has focus). It also gives you a little interface to inspect the pixel values of the image at particular position which might be helpful.

1.4 Processing the image

After having dealt with the variable declaration, input/output matters, one can actually start processing the image. The dimensions of the image can be accessed by

```
/* .. .. .. do some processing .. .. .. */
unsigned width = myFloatImg.dimx();   // access image width
unsigned height = myFloatImg.dimy();  // access image height
```

Sometimes it is necessary to initialize the image, such that all pixels have a particular value. This can be achieved by the function call

```
myFloatImg.fill(0.0f);
```

which, in this example, sets every pixel to zero.

As already mentioned, the access of the pixels is intuitively like one would access a 2-dimensional array, i.e.

```
for (unsigned y=0; y < height; y++) {
  for (unsigned x=0; x < width; x++) {
    myFloatImg(x,y) = x*y;
  }
}
```

1.5 Compilation and Linking

For convenience, we provide a simple script which deals with all matters of compiling and linking a source-code to an executable. This is achieved by invoking the command

```
biwicompile  yourSourceFile.cc
```

where the argument obviously is the file containing your code. The resulting binary will have the same name, but without the extension, in this example it would be "yourSourceFile".

2 Argument Reading using Argstream

"argstream.h" is a c++-template based library designed for simple command line argument reading. Its usage is very simple and is explained by the following code snippet:

```
 1  #include "argstream.h"
 2
 3  int main(int argc, char**argv) {
 5    argstream as(argc, argv);
 6
 7    string sFilename="defaultFile.png";
 8    int    nNumber=3;
 9    float  dFloat;
10    as >> parameter("i", sFilename, "Path to a local file ", true);
11    as >> parameter("n", nNumber, "some number ", false);
12    as >> parameter("d", dFloat, "a floating point", false);
13
14    as >> help();
15    as.defaultErrorHandling();
16    return 0;
17  }
```

Firstly, to use the library one has to include its header file 'argstream.h' (line 1). Line 5 defines an argstream object and the lines 14 and 15 deal with the error handling and displaying of an automatically generated usage text which is displayed when the program is called with the option – –help. This lines will always be the same and you do not have to care about them; just copy them as they are!

The interesting part is in the middle, i.e. lines 7 to 12. Basically, for every command line option you have to declare a variable. You may already initialize it with some value (which will result in a default value for that option). Then, the actual argument reading is achieved by the lines 10..12. The syntax is identical for all types of variables you'd like to read from cmdline and is simply

```
as >> parameter (<Name: string>,
                 <Var: variable of some type>,
                 <Description: String>,
                 <Mandatory?: bool>);
```

where the first argument indicates the option name (which will be given on the command line, eg. – –Name value) the second specifies the variable where the value provided on the command line has to be stored. The third argument gives a short description about the option (which is displayed when giving the option –help) and the last argument indicates whether the user always has to provide that command line option, i.e. it is mandatory, or if it is optional. In case of an optional variable you have to put 'false' and if it s mandatory, the argument has to be set to 'true'.

3 References

Further, more detailed information for the two libraries can be found on the web at the following two web sites:

- http://cimg.sourceforge.net/
- http://artis.imag.fr/ Xavier.Decoret/resources/argstream/

4 Sample Program

```
/*****************************************************************
 * warp.cc
 * compile with:
 *   biwicompile warp.cc
 * execute the program (example):
 *   warp --i ~cvcourse/pic/stpeter.rgi --o warped.rgi --strength 5.3
 * Strength should be a value between 0 and 16 (default: 8).
 *****************************************************************/
#include <string>
#include <iostream>
#include "argstream.h"     // command line parameter reading
#include "CImg.h"          // C++ image library
using namespace cimg_library;
using namespace std;

int main(int argc, char **argv) {
  argstream as(argc, argv);

  string infile="",
         outfile="out.rgi";
  float  strength=8.0f;
  as >> parameter ("i", infile, "input file name", true);
  as >> parameter ("o", outfile, "output file name", false);
  as >> parameter ("strength", strength, "warp strength", false);
  as >> help();
  as.defaultErrorHandling();

  CImg<float> in(infile.c_str());
  const int width = in.dimx();
  const int height = in.dimy();
  if ((width==0) || (height==0)) {
    cerr << "Error when loading image." << endl;
    return -1;
  }
  in = in.get_norm_pointwise(1)/(float)in.dimv(); // ensure greyscale img!

  // warp the image
  CImg<float> out(in); out.fill(0.0);
  const int xmax = width  - 1;
  const int ymax = height - 1;
  for (int y = 0; y < height; y++)
    for (int x = 0; x < width; x++) {
      float weight = strength*x*(xmax-x)*y*(ymax-y)/(xmax*xmax)/(ymax*ymax);
      int new_x  = (int) ((1-weight)*x + weight*      y * xmax/ymax);
      int new_y  = (int) ((1-weight)*y + weight*(xmax-x)* ymax/xmax);
      out(x,y) = in(new_x,new_y);
    }

  // write the output image
  out.save(outfile.c_str());
  out.display("warped");

  return 0;
}
```