Introduction to CImg: Comments

April 1, 2009

This paper presents some mistakes that were often done, along with some improvements that could be done. They are presented in no particular order. Given remarks are not made to criticize anyone. Only mistakes that were often done will be presented.

1 Using references

When giving images as parameters, or any kind of object of some size, it is better to use references. References prevent copy of objects by giving the address where the object may be found. References are noted by adding &.

Let us consider the 2 following examples. In both cases the same result will be obtained, however the first case will be slower and use more memory. In the first case a local copy of the image will be done for the method get_max. In the second case the location inside memory for the image will be given.

```
float get_max(CImg<float> img);
float get_max(CImg<float> &img);

void main()
{
 CImg<float> img("toto.png");
 filter(img);
}
```

However you should be careful using references, because any modification of the parameter img will be kept inside the main function for the second case, while without references changes are only local. You may note that using pointers will lead to similar results.

2 Method pow inside math

When computing square values many persons used the pow method. You should avoid that. This method applies an algorithm that converges to the power function. It will be slower than a multiplication and may be less precise.

3 Declaration of global variables

Variables should be declared only inside methods. When you are doing it outside a method, they become global. It will create lots of problems when you merge two projects that uses global variables with same name.

```
CImg<float> img("toto.png");    // defining a global variable

void main()
{
    img.blur(2.0f);
}
```

How it should be done:

```
void main()
{
    CImg<float> img("toto.png");    // defining a local variable
    img.blur(2.0f);
}
```

4 Use of radius

Using a radius to define the size of a Gaussian mask as $(2 \times radius + 1)$ allows to avoid even sizes of kernels. Such kernels will give some results, but it will not be interesting in our case. The value of the Gaussian distribution quickly decreases when moving away from the center of the kernel. Thus the pixel with the most weight will be the one in center of the kernel, which requires an odd size of kernel.

5 Normalization of the Gaussian kernel

The integral of a Gaussian distribution is 1. The value of the Gaussian distribution quickly decreases when moving away from the center of the kernel. Thus, when we are far enough compared to the standard deviation of the distribution, the values become non significant. It permits to consider only a small neighborhood to apply a Gaussian kernel.

However, when the considered neighborhood is too small, the norm of the Gaussian kernel becomes lesser than 1. This leads to a rescaling of the intensities of the image after convolution. This is easily seen, because we obtain darker images than the initial ones. To avoid that, one only has to normalize the kernel, meaning to multiply the kernel values in order to obtain a sum of the terms of 1.

6 Doing a loop on a 2D image

In general keeping things simple is always better. While doing a loop on an image, it is simpler to do 2 loops than using quotient and remainder of euclidean division. How it should be done:

```
CImg<float> Kernel(int radius, float sigma)
{
    int size = 2*radius+1;

    CImg<float> kernel(size, size);

    for(int y=0 ; y<size ; y++)
    {
        for(int x=0 ; x<size ; x++)
            kernel(x,y) = ...; // compute the value here
    }

    kernel = kernel/kernel.sum();
    return kernel;
}
```

What you should avoid:

```
CImg<float> Kernel(int radius, float sigma)
{
    int x,y;
    int size = 2*radius+1;

    CImg<float> kernel(size, size);

    for(int i=0 ; i<size*size ; i++)
    {
        x = i%L;
        y = i/L;
        kernel(x,y) = ...; // compute the value here
    }

    kernel = kernel/kernel.sum();
    return kernel;
}
```

7 Default value

When the prototype of a method is given, it may specify some default values. It gives the value the parameter will take if no value is given by the programmer. Parameters with default values are always given after the mandatory parameters. The order is important, because you cannot use the default value of the parameter before the last without using the default value of the last parameter too.

For example, in the case of the noise method inside CImg, the 2 given call to the noise methods are equivalent.

```
CImg<T>& noise(const double sigma, const unsigned int noise_type=0);

img.noise(2.0f, 0);
img.noise(2.0f);
```