

Image Processing Library C++ Edition v1.2

Introduction, Tutorial, and User Guide

Qi Zhao

University of Illinois, Urbana-Champaign

Web: <http://www.ifp.uiuc.edu/~qizhao>

Email: qizhao@ifp.uiuc.edu

copyright reserved

November 1, 2002

1 INTRODUCTION

1.1 Library Overview

IPL/C++ (image processing library C++ edition) is a result of two things: the need of useful common image processing routines, and the frustration of not being able to find a satisfactory and free one.

As an engineering student working in the image processing area, I have to write a lot of small image processing related routines for research purposes, things such as image rotation, resizing, resampling, histogram equalization. Of course we can use free/commercial programs to do it, but they are standalone programs which you cannot embed easily in your simulation code. Using simulation software packages like Matlab is convenient but inefficient. So what we need is a set of C or C++ routines which can be called as a library.

IPL is not for practical real-time usage. It's not optimized for speed (although I do optimize my code when possible). It's optimized for ease of use and ease of being extended. It provides you with quick-and-dirty routines to perform what you need in your own simulation code. It is written for students and researchers who need image manipulation functionality but do not have time to write the code.

IPL/C++, as the name implies, is written in ANSI C++. I enclosed everything in a single class: **Cimage**. It contains a private RGB buffer to store the image, and has many methods which can be used to perform all sorts of image processing operations. The complete feature list includes:

1. **image conversion:** RGB \leftrightarrow grayscale, RGB \leftrightarrow HSI, RGB \leftrightarrow YCrCb.
2. **chroma manipulation:** setting/adjusting hue and saturation, solid fill.
3. **lumina manipulation:** setting/adjusting brightness and contrast, auto-contrast.
4. **geometric manipulation:** free-angle rotation, resizing.
5. **multiple format support:** importing/exporting to raw, pbm, pgm, ppm, gif (read only), and (progressive) jpeg format.

The included demo program illustrates most of the functionalities of IPL/C++.

1.2 Supported Platforms

IPL/C++ is written in ANSI C++, so it should compile and run on almost all systems that have an ANSI C++ compiler on it. I have personally tested it on Linux, Solaris and Win32. However, I would like to point out that because IPL/C++ relies its multiple image-formats support on external libraries, whose behavior in a Windows system is unpredictable at best, certain errors might occur when you try to use it on a Windows platform.

1.3 Software License

IPL/C++ is distributed free of charge, in the form of source code. It is released as OpenSource software under GPL 2.0. You are free to download, use and modify them, so long as you conform to the specifications in the GPL license, a copy of the license is included in the distribution.

1.4 Availability and Installation

You can download the source code and binary library packages of IPL/C++ from my homepage at <http://www.ifp.uiuc.edu/~qizhao>. Updates and fixes would also be posted to my homepage regularly.

IPL/C++ utilize the independent JPEG group's libjpeg for JPEG input/output support, and libungif for GIF input support. **You need to have these two libraries installed in order to have JPEG or GIF support.** To disable them, you can manually edit the `Makefile` and remove or comment out the two lines related with JPEG and GIF support.

Because the way the JPEG library header file `libjpeg.h` was written, you need to modify it before you can call its C library from inside C++ programs. Modified versions of `libjpeg.h` and `jmorcfig.h` are included in the distribution, you need to put them in the same directory as the IPL/C++ source code. There is no such problem with the header file for GIF library.

For installation, check the following platform specific instructions:

1. **Windows:** if you are using the cygwin GCC compiler, you can simply type in "make" to compile the library. If you are using the Microsoft Visual C/C++ compiler, first make sure that you have run "vcvars32.bat" from your VC bin directory (only if you are running Windows 95/98/ME, not for Windows NT/2K), then type in "nmake /f Makefile.vc" to compile.
2. **UNIX/Linux:** run "make" and "make install".

1.5 Author's Disclaimer

I, as the provider of the source code, do not take any responsibility for whatever might happen when you use them. You are using them at your own risk. I do not provide any form of support for these codes, but I will try to fix bugs reported to me and improve efficiency of the codes.

2 TUTORIAL

The included `demo.cpp` program serves as the best tutorial for writing code with IPL/C++. For a simpler example, look at the following code:

```
001: #include "imglib.h"
...
010: int main(int argc, char *argv[]) {
...
015:     Cimage img;
016:     Cimage *newImg;
017:     img.input("test.jpg", JPG);
018:     newImg = img.rotateAngle(130);
019:     newImg.output("test.ppm", PPM);
020:     delete newImg;
...
}
```

This is probably the simplest usage of IPL/C++. Let us take a closer look at these lines carefully:

- 001. This is the mandatory line to use IPL.
- 015. Define a `Cimage` object `img`.
- 016. Declare a pointer `newImg` to a `Cimage` object.
- 017. Read a jpeg image file. Note that memory to store the image is not allocated until after this operation, and will be freed when `img` goes out of scope. Currently the supported file formats are: GIF (read only), JPEG, RAW, PBM, PGM, and PPM.
- 018. Rotate the image by 130 degrees and put the resulting (larger) image in `newImg`, memory is automatically allocated but needs to be manually freed by calling `delete newImg`. A new `Cimage` object is created whenever the image processing operation generated a different sized image from the original. For a detailed list for all available operations, please refer to Section 3.
- 019. Output the new image to a PPM format file.
- 020. Free the memory allocated for the new image.

As you can see, IPL/C++ is fairly straightforward to use. You need at least the following files to use IPL: `libipl.a`, `imglib.h`. To use JPEG and GIF support, you need to have the C version of `libjpeg` and `libungif` installed, plus the helper class header files `Cjpeg.h`, `Cgif.h`, and the modified JPEG library header file `jmorecfg.h`.

3 USER GUIDE

3.1 Cimage Class Internals

`Cimage` class maintains three independent dynamically created buffer to store image data: the RGB buffer, YCrCb buffer, and HSI buffer. The RGB buffer is also used for input/output operations, i.e, data must be converted and stored to RGB buffer before output methods are called, this step is done by supplying a "TRUE" flag to some of the image processing routines. This flag can be considered as the "Final Step" step, For example:

```
...
001: img.adjustHue(150);
002: img.setSaturation(120);
003: img.autoContrast(TRUE);
004: img.output("final.jpg", JPG);
...
```

As shown above, the "TRUE" flag is not supplied until the last image processing operation, after which it is ready for output. Note that if you do not supply the flag, it is default to "FALSE".

RGB, HSI, and YCrCb buffers are exposed in the API, be careful what you do with them! My recommendation is always to leave them alone and use the image processing functions to operate on them. If you really need a specific function, obtain the source code and write an extension.

To input/output JPEG image files, a helper class `Cjpeg` is created, it can also be separately used outside of `Cimage` class if you need to access jpeg image files.

To input GIF image files, a helper class `Cgif` is created. It can also be used separately outside of the `Cimage` class if you need to access gif image files.

3.2 Alphabetic API List

3.2.1 Cimage class methods

- `void adjustBrightness(double factor)`
Adjusting brightness based on current value. `factor` is a percent value, i.e, `factor = 150` means increaseing current brightness to 150% of the original value.
- `void adjustContrast(int factor, bool toRGB=false)`
Adjusting the contrast. `factor` is a percent value. The `toRGB` flag is the "Final Step" flag mentioned in the previous section. If this function is the last image operation called before `output()`, then `toRGB` should be set `TRUE`, otherwise it should be set `FALSE` to be more efficient.
- `void adjustHue(double degree, bool toRGB=FALSE)`
Adjusting hue based on current value. `degree` has a value between 0 and 360.
- `void adjustSaturation(double factor, bool toRGB=FALSE)`
Adjusting the saturation based on current value, `factor` is a percent value.
- `void autoContrast(bool toRGB=FALSE)`
Automatically adjust contrast. It is equivalent to histogram equalization.

- `void cropRect(uint x, uint y, uint width, uint height)`
Crop the image using a rectangular window specified using the top-left coordinate (x, y) and the window size (width, height).
 - `void fillSolid(Color fillColor)`
Fill the image with solid color. `Color` is a structure with three components R, G and B.
 - `unsigned int getCol(void)`
Get the width of the image. It can be used after the `input()` operation.
 - `unsigned int getRow(void)`
Get the height of the image.
 - `TYPE getType(void)`
Get the type of the image. `TYPE` can be one of the enumerated values: `GRAY`, `RGB`, and `YUV`.
 - `void histogram(double *x, ulong xSize, unsigned int *hist, unsigned int numBin)`
Calculate the histogram of array `x` of size `xSize`. The result is returned in the array `hist` of size `numBin`.
 - `void HSI_RGB(void)`
Convert the HSI buffer to RGB buffer.
 - `void input(char *inFile, FORMAT f)`
Input image with specified format. `FORMAT` can be one of the enumerated values: `RAW`, `PBM`, `PGM`, `PPM`, `GIF`, and `JPG` (you can also use all lower-case `pgm` and `ppm` to indicate the ASCII format variant of `PGM` and `PPM`).
 - `void mirrorHorizontal()`
Flip the image using a vertical axis (horizontal mirroring).
 - `void mirrorVertical()`
Flip the image using a horizontal axis (vertical mirroring).
 - `void output(char *inFile, FORMAT f)`
Output image with specified format. `FORMAT` can be one of the enumerated values: `RAW`, `PGM`, `PPM`, and `JPG`. Note that you need to call the `setSize()` function before the `output()` function if the output format is `RAW`. If the output format is `JPG`, you can optionally call `setJpgProperty()` to specify the JPEG quality factor or create a progressive JPEG file.
- `void RGBtoGray(void)`
Convert color image to grayscale.
- `Cimage* resize(double rowFactor, double colFactor, METHOD method=SPLINE)`
Resize the image using one of the specified methods. Both `rowFactor` and `colFactor` are percent values, `METHOD` is one of the following enumerated values: `SINC` or `SPLINE`. Basically `SINC` method is slower but does a better job, `SPLINE` method is faster and gives reasonable quality.
 - `Cimage* resize(unsigned int newRow, unsigned int newCol, METHOD method=SPLINE)`
Resize the image to the specified height and width.

- `void RGB_HSI(void)`
Convert the RGB buffer to HSI buffer.
- `void RGB_YCrCb(void)`
Convert the RGB buffer to YCrCb buffer.
- `void reduceColorDepth(unsigned int numBits=8)`
Reduce the color depth of the current image to specified bits. 8-bit stands for 256 colors, 24-bit stands for true color.
- `void rotateAngle(int angle)`
Rotate the image by a degree that is a multiple of 90, such as 180, 270, etc.
- `Cimage* rotateAngle(double degree)`
Rotate the image by any degree. Note that this operation increases the size of the image, thus a new `Cimage` object is created to store the new image.
- `void setHue(double hue, bool toRGB=FALSE)`
Set a uniform value for hue.
- `void setJpgProperty(unsigned int quality, int progressive=FALSE)`
Set the quality factor and progressive flag for outputting JPEG files. `quality` is an integer between 1 and 100, with 100 gives the best quality but least compression. Setting `progressive` to `TRUE` generates progressive JPEG files suitable for webpages. **This function is available only when compiled with JPEG support.**
- `void setSaturation(double sat, bool toRGB=FALSE)`
Set a uniform value for saturation.
- `void setSize(unsigned int row, unsigned int col)`
Set the image width and height to specified values. It should be called before inputting a RAW format image or create a new image.
- `void setType(TYPE type)`
Set the image type to specified value.
- `void YCrCb_RGB(void)`
Convert the YCrCb buffer to RGB buffer.

3.2.2 Cjpeg class methods

These methods are available only when JPEG support is enabled in compile time.

- `int exportRGB(double *R, double *G, double *B)`
Export the imported JPEG image to external RGB buffers, must be called after `readJPEG()`.
- `unsigned int getColorChannel()`
Obtain the color information of the current JPEG image, must be called after `readJPEG()`. “1” means grayscale image, “3” means RGB image.
- `unsigned int getFrameSize()`
Obtain the dimension of the current JPEG image, must be called after `readJPEG()`.

- `unsigned int getWidth()`
Obtain the width of the current JPEG image, must be called after `readJPEG()`.
- `unsigned int getHeight()`
Obtain the height of the current JPEG image, must be called after `readJPEG()`.
- `unsigned int getImgSize()`
Obtain the memory buffer size of the current JPEG image, must be called after `readJPEG()`. It is equal to the product of image dimension and number of colors.
- `int importRGB(double *R, double *G, double *B)`
Import the content of the external buffer into the JPEG image.
- `int readJPEG(char *jpgFileName)`
Read the specified JPEG image file into memory, this function must be called before all other Cjpeg class methods.
- `void setImgSize(unsigned int width, unsigned int height, unsigned int colorChannel)`
Set the dimension and color information of the output JPEG image.
- `void setQuality(unsigned int quality)`
Set the quality factor of the output JPEG image.
- `void setProgressive()`
Set the output JPEG image to be progressive.
- `void setRestartMarkers(int i)`
Set the restart marker interval of the JPEG file (if you do not know what this means, leave it alone).
- `int writeJPEG(char *jpgFileName)`
Output the JPEG image.

3.2.3 Cgif class methods

A GIF file can contain several images on a virtual “screen” with certain width and height. Each image has its own width and height, plus a top-left coordinate to the screen. GIF images are *indexed* images, meaning that pixel color information are stored as index values into a *colormap*, instead of the RGB values themselves. GIF images are limited to a maximum of 256 colors.

GIF output function is currently not supported because of the famous license issue with Unisys. I might consider adding it when I find a way to circumvent the patent issues.

These methods are available only when GIF support is enabled in compile time.

- `int exportRGB(double *R, double *G, double *B, int imgID=0)`
Export the specified GIF image to external RGB buffer. You can use the `imgID` variable to specify which image to extract. This function must be called after `readGIF()`.
- `ColorMapObject *getColorMap(int imgID=0)`
Get the colormap of the specified image in the gif file.
- `unsigned int getColorResolution()`
Get the color resolution of the GIF image.

- `ColorMapObject *getGlobalColorMap()`
Read the global colormap of the GIF image file, this is the fall-back colormap if not image-specific colormap is found.
- `unsigned int getImageHeight(int imgID=0)`
Get the height of the specified image.
- `int getImageLeftCord(int imgID=0)`
Get the left coordinate of the specified image.
- `int getImageTopCord(int imgID=0)`
Get the top coordinate of the specified image.
- `unsigned int getImgCount()`
Get the number of images stored in the current GIF file.
- `int getScreenBgColor()`
Get the screen background color of the GIF file. Note that this is an **index** into the colormap, not an actual RGB color.
- `unsigned int getScreenHeight()`
Get the screen height of the current GIF file.
- `unsigned int getScreenWidth()`
Get the screen width of the current GIF file.

4 ToDo List

More Image Formats: TIFF, PNG, BMP, XPM format support are planned.

Intelligent Processing: auto-balance, auto-contrast, auto-hue, auto-saturation, etc.

GUI Front-end: IPL/C++ is designed to be a programming library only, but it might help to have a simple GUI front-end to demonstrate its ability. I plan to use Tcl for the GUI.

Scripting Engine: a scripting engine takes input of an image processing script and process everything in one run.

5 Appendix: Revision History

- 09/01/01:** Initial release version 1.0.
- 11/01/01:** JPEG read/write support added.
- 11/09/01:** Demonstration program added.
- 11/10/01:** Bug fix and release version 1.1.
- 11/12/01:** GIF read support added.
- 11/15/01:** JPEG and GIF support now optional in compilation.
- 11/16/01:** Added PBM and ASCII format PGM/PPM support.
- 11/18/01:** Added horizontal and vertical mirror support.
- 11/20/01:** Bug fixes for core-dump and rotation.
- 11/26/01:** Added cropping support.
- 11/27/01:** Bug fixes and release version 1.2.
- 01/15/02:** small bug fixes and minor release version 1.2.1.