OPENCV TUTORIAL: IMAGE PROCESSING

INTRODUCTION

What is OpenCV? This might be the 'basic' question that comes first to your mind. Well, it stands for **'Open Source Computer Vision Library'** initiated by some enthusiast coders in '1999' to incorporate Image Processing into a wide variety of coding languages. It has C++, C, and Python interfaces running on Windows, Linux, Android and Mac.

Before, I get into the use of OpenCV; Let's get familiar with the same. As, by now you must have understood that it is a **'library'**, so to use the functions available in the library you would need a compiler.

To start off, you need to install **'OpenCV'** and a **'compiler'** and then establish a linkage between the two (i.e. Compiler is able to use the functions available with the library).

Getting Started:

OpenCV can be downloaded from the following link: http://sourceforge.net/projects/opencvlibrary/

Choose any of the several available versions. Prefer OpenCV2.1, if interested in simple image processing (we have used that version while preparing this tutorial).

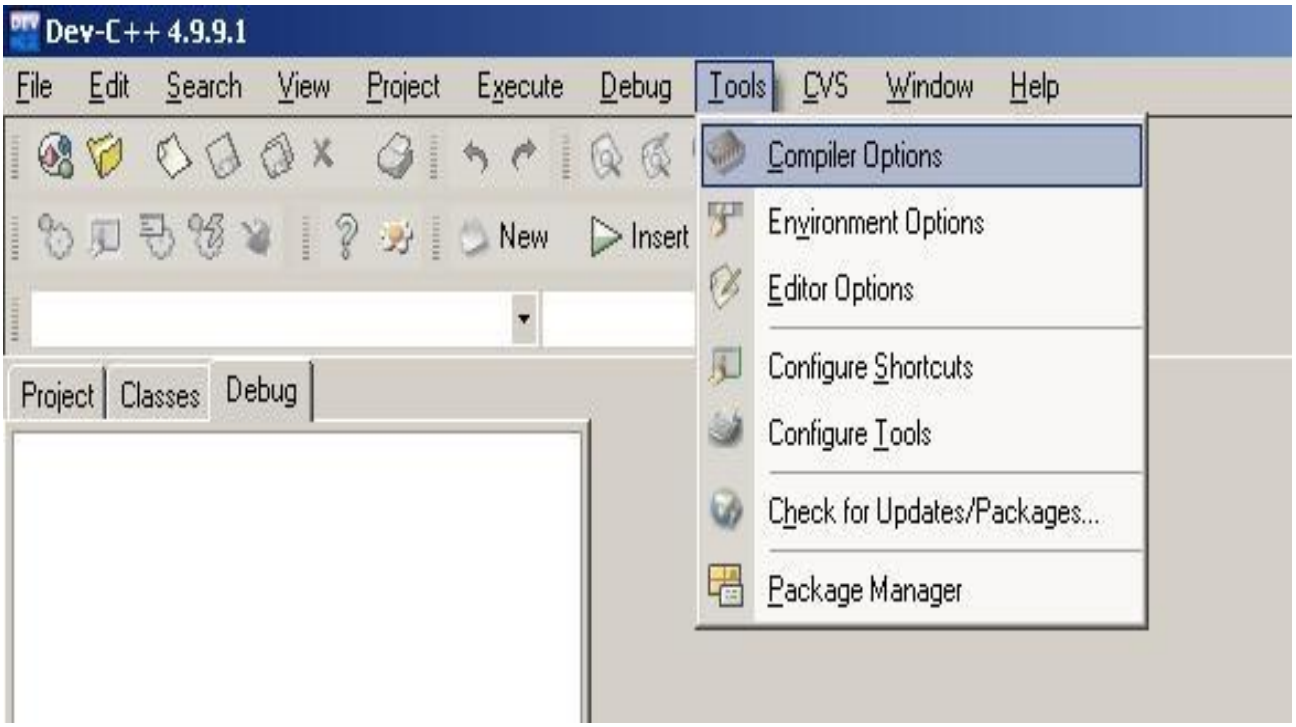
Next, you would need a compiler like DevC++, CodeBlocks, VisualC++. These can be downloaded from the following links:

- DevC++ : http://sourceforge.net/projects/dev-cpp/files/Binaries/Dev-C%2B%2B%204.9.9.2/devcpp-4.9.9.2_setup.exe/download
- CodeBlocks : http://www.codeblocks.org/downloads/26/
- VisualC++ : http://www.microsoft.com/visualstudio/en-us/products/2010-editions/visual-cpp-express

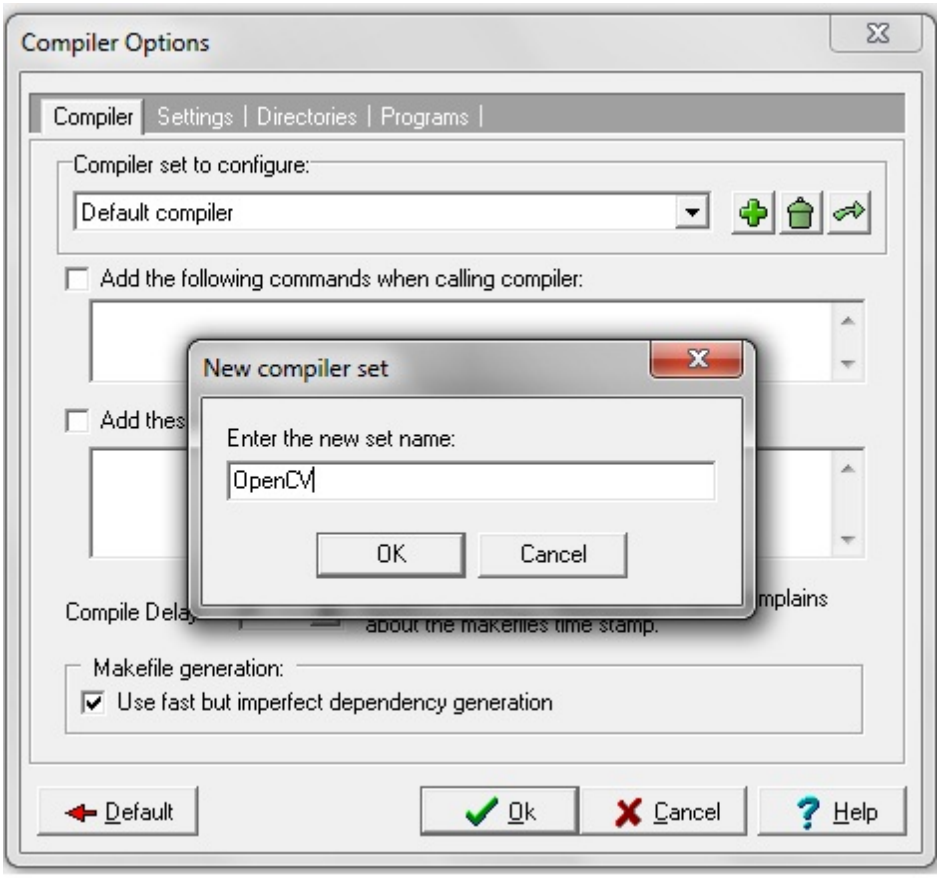
Install one of the above compilers and next you would need to **link** the library with the installed compiler.

For integrating OpenCV with DevC++ :

- First of all, you have to indicate the header files you want to add. For that, select ***Tools->Compiler Options*.**



- Then click on the **plus** sign(under **compiler set to configure**) to add a new compiler named here, **OpenCV*.***



- To finish on, in the section **Add the following commands..** write

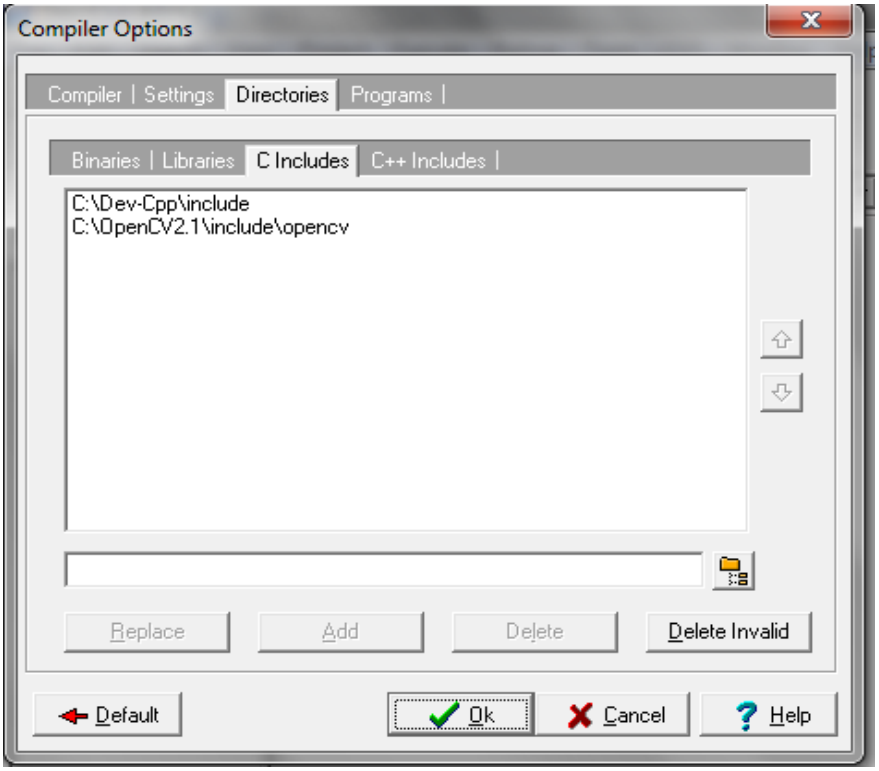
  -L"C:\OpenCV2.1\lib" -lcxcore210 -lcv210 -lcvaux210 -lhighgui210 -lml210

  [note that the text in inverted commas is basically the location of the lib(libraries) folder of OpenCV2.1. So, it won't work well if u've not installed OpenCV in the default folder.]

Configuring included files

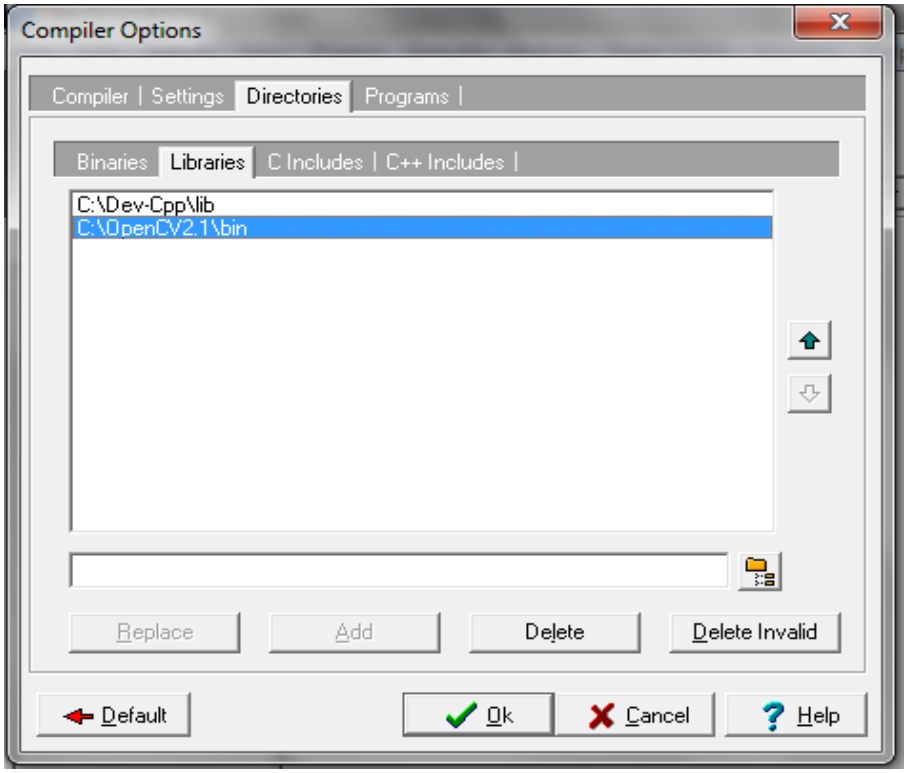
Next, click on *Directories* and then on *C Includes* to add all the headers, located in some *C:\OpenCV2.1* subdirectories. You only need to add ***C:\OpenCV2.1\include\opencv*** in the include tab to get things to work**.**

If you want to code in C++ then do the same for C++ includes



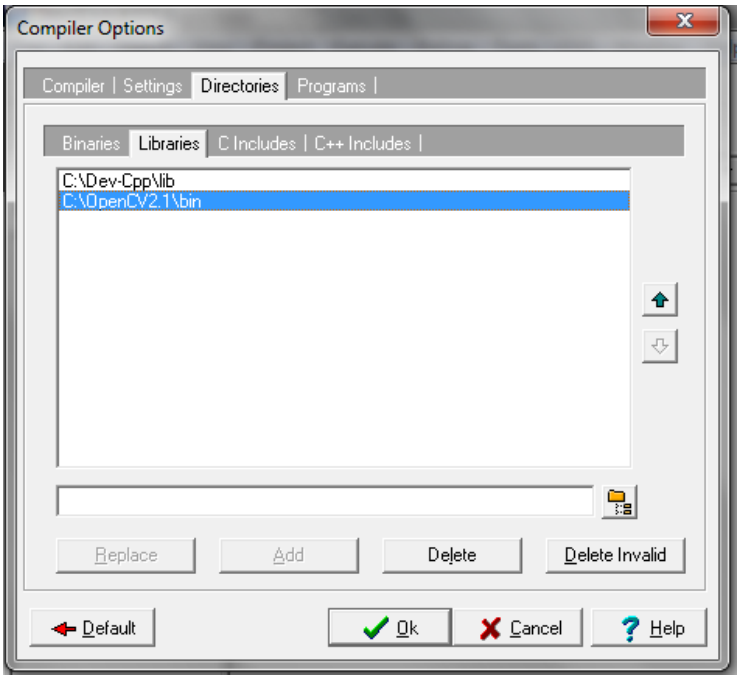
Configuring static library files

In the *libraries* section under the same heading directories you will need to add ***C:\ OpenCV2.1\lib*** **.** (if already present, ignore this step.)



Configuring dynamic library files

And to finish, add the bin directory where the dlls are:
i.e. add ***C:\ OpenCV2.1\bin*** to *binaries* subdivision.



Congratulations..!!

With this, You are done with configuring OpenCV with DevCPP, you can try running a sample code.

OpenCV can also be configured with CodeBlocks and VisualC++ by following the instructions on provided link

CodeBlocks: http://opencv.willowgarage.com/wiki/MinGW

VisualC++: http://www.scribd.com/doc/60304851/Steps-to-Integrate-Opencv-2-2-With-Visual-Studio-2010

By now, you must be having a compiler integrated with OpenCV library. Before I move to next section in which I will take you through basic image processing let me introduce you to the basic OpenCV modules.

Basically there are four modules. I'll explain briefly about each module.

- **cv:** Main OpenCV functions, Image processing and vision algorithms.
- **cvaux:** Auxiliary (experimental) OpenCV functions.
- **cxcore:** Data structures, linear algebra support, XML support drawing functions and other algorithms.
- **highgui:** GUI functions, Image and Video I/O.

Depending on what your program implements you wish to use, you should include corresponding modules.

Image Processing

“Image Processing” is what it intuitively suggests.

Image processing is **IMAGE + PROCESSING** .

What is an Image?

- Image is a collection of **PIXELS.**
- Each pixel is like an array of numbers.
- These numbers determine the color of the pixel.

Now let me introduce you to different types of images. There are three types of images:

Binary image,

Grayscale Image and,

Coloured image.

Each kind of image has few attributes attached to it like **number of channels** and **depth .**

- Number of channels : Defines the dimension of array , each pixel is.
- Depth : Defines the maximum bit size of the number which is stored in the array

Let’s have a closer look at different types of images.

Binary Image

Again, as the name suggest each number associated with the pixel can have just one of the two possible values.

- Each pixel is a 1 bit number.
- It can take either 0 or 1 as its value.
- 0 corresponds to **Black**
- 1 corresponds to **White**
- Number of channels of a binary Image is 1
- Depth of a binary image is 1(bit)

Example of a binary image

Grayscale Image

- Each pixel is a 8 bit number
- It can take values from 0-255
- Each value corresponds to a shade between black and white(0 for black and 255 for white)
- Number of channels for a grayscale image is 1
- Depth of a gray scale image is 8(bits)

Example of a grayscale image

RGB Image

- Each pixel stores three values:
  1. R : 0-255
  2. G : 0-255
  3. B : 0-255
- Each number between 0-255 corresponds to a shade of corresponding color
- Depth of a RGB image is 8(bits)
- Number of channels for a RGB image is 3

Example of a RGB image

Starting with Processing Images

I will be using C language and DevC++ as compiler embedded with OpenCV2.1. You need not worry if you are using a different compiler.

1. Displaying an image

To start, let's get through a simple program of displaying an image already saved on the disk (something similar to a 'hello world type program').

Every C code starts with inclusion of header file and this is nothing different. So, we will include basic header files needed for our program.

We will need following header files:

- **cv.h**
- **highgui.h**

Image is stored as a structure IplImage *with following elements.*

```
int  nChannels;     // Number of color channels (1,2,3,4)
int  depth;         // Pixel depth in bits:
                    //   IPL_DEPTH_8U, IPL_DEPTH_8S,
                    //   IPL_DEPTH_16U,IPL_DEPTH_16S,
                    //   IPL_DEPTH_32S,IPL_DEPTH_32F,
                    //   IPL_DEPTH_64F
int  width;         // image width in pixels
int  height;        // image height in pixels
char* imageData;    // pointer to aligned image data
                    // Note that color images are stored in BGR order
int  dataOrder;     // 0 - interleaved color channels,
                    // 1 - separate color channels
                    // cvCreateImage can only create interleaved images
int  origin;        // 0 - top-left origin,
                    // 1 - bottom-left origin (Windows bitmaps style)
int  widthStep;     // size of aligned image row in bytes
int  imageSize;     // image data size in bytes = height*widthStep
struct _IplROI *roi;// image ROI. when not NULL specifies image
                    // region  to be processed.
char *imageDataOrigin; // pointer to the unaligned origin of image data
                       // (needed for correct image deallocation)

int  align;         // Alignment of image rows: 4 or 8 byte alignment
                    // OpenCV ignores this and uses widthStep instead
char colorModel[4]; // Color model - ignored by OpenCV
```

There is no need to go into the details right now, we will get acquainted with these elements during the course of the tutorial.

Steps involved:

We first need to initialize a pointer to the image (structure)

IplImage * input;

Next, load the desired image to the pointer

input = cvLoadImage(“filename.extension”,1);

[1 for colored and 0 for grayscale]

Note: The image must be stored in the same folder in which you save the C program.

To display the loaded image you will need a window.

cvNamedWindow (“window name”, 1);

Again [1 for colored and 0 for grayscale]

Above command creates a window and to display the loaded image we use the following command:

cvShowImage(“window name”, input);

Suppose you have an image named “**shivendu.jpg**” in the same folder in which you save your code then your code would look something similar to the following

```
#inclde “cv.h”

#include “ highgui.h”

Int main()

{

 IplImage * input;

 input = cvLoadImage(“shivendu.jpg”,1);

 cvNamedWindow(“Output”,1);

 cvShowImage(“Output”, input);

}
```

If you try to execute this code the output window flickers just once. To appreciate the output we need **cvWaitKey(number)**

- If 0 or negative number is given as input: - Waits indefinitely till key press and returns the ASCII value of the key pressed.
- If positive number is given as input: - Waits for corresponding milliseconds.

Now the final code looks like

```
#inclde “cv.h”                          // Include header files
#include “ highgui.h”
Int main()
{
 IplImage * input;                  // Variable declaration
 input = cvLoadImage(“shivendu.jpg”,1);    // Loads the image
cvNamedWindow(“Output”,1);                 // Creates a window to display image
 cvShowImage(“Output”, input);           // Displays the image
cvWaitKey(0);                                  // Waits till a key  is pressed
 }
```

This simple code must have helped you in understanding the flow of the program (This is how things work with OpenCV). A good programmer will always clear the memory assigned to variables.

It is therefore advisable to release the image and destroy the window created:

```
cvDestroyWindow( "Output" );        //destroy the window
cvReleaseImage( &input );           //release the memory for the image
```

Include above two lines to make it a good simple code.

1. **Creating an image**

To create an image you need to provide the following details

- Size(height and width)
- Depth
- Number of channels
- And specify the pixel values

For creating an image we need we use the following function:

output=cvCreateImage(cvGetSize(input),IPL_DEPTH_8U,3)

This will create a RGB image(most general case among the three types of images discussed) without specifying pixel values

2. **Some common OpenCV functions**

- output=cvCloneImage(input)
  -----Copies image from input to output
- cvCvtColor(input, output, conversion type)
  { Conv. type : CV_BGR2GRAY ,CV_BGR2HSV}
  -----Saves input image in output pointer in other color space
- cvSaveImage("output.jpg",output)
  -----Saves image pointed by output naming it output

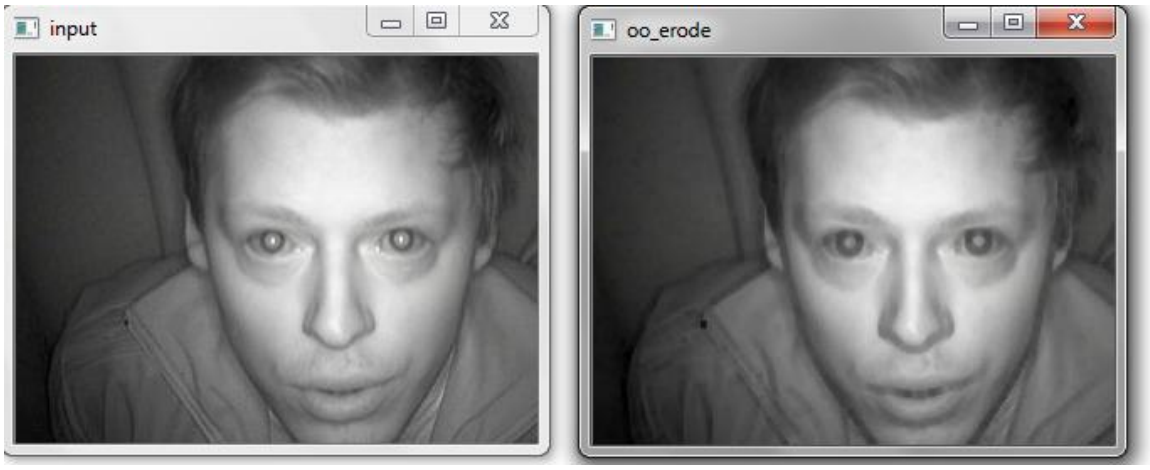
3. **Morphological operations on a image**

There are two different kinds of morphological operations :

1. Erosion
2. Dilation

For carrying out morphological operations we need to specify type of structural element and number of iterations.

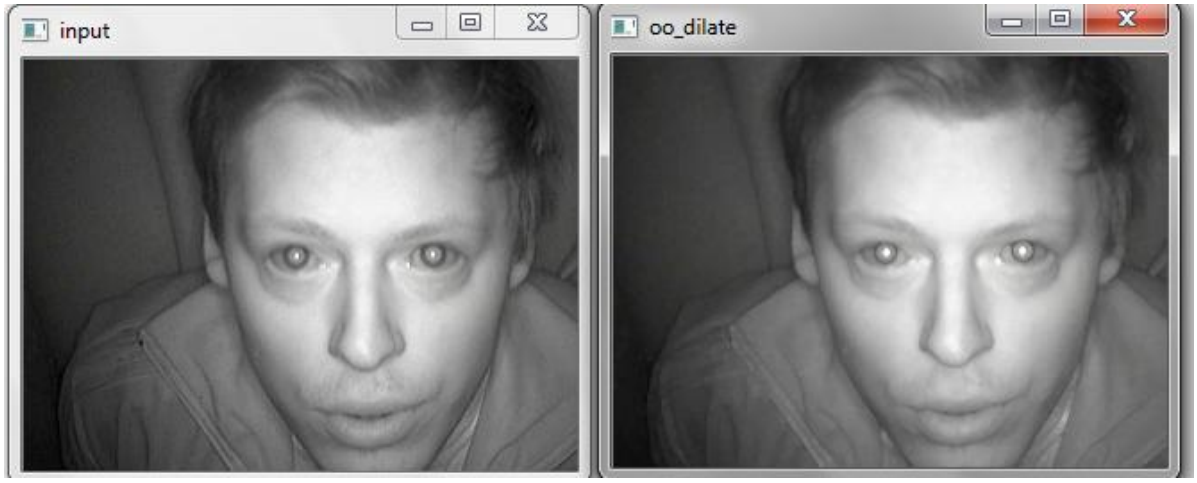
Erosion erodes the image. It tries to bring uniformity in the image by converting bright points in neighborhood of points of less intensity into darker ones



Input Image Eroded image

Notice the change in eyes, illuminates spots in the eyes are removed because in the input image there is a stark change in illumination at points near pupil.

Dilation dilates the image. It tries to bring uniformity in image by converting dark points in neighborhood of points of higher intensity into bright ones



Input Image Dilated iamge

Here, is the code which erodes and dilates an image saved in the same folder where c code is saved

```
#include "cxcore.h"

#include "highgui.h"

#include<cv.h>

int main()

{

int i=1;

IplImage* input;

IplImage* dilate;

IplImage* erode;

IplConvKernel *structure_element;

structure_element=cvCreateStructuringElementEx(i*2+1, i*2+1, i,i,CV_SHAPE_ELLIPSE );     // Defines the structural element

cvNamedWindow("ii", 1);

cvNamedWindow("oo_dilate",1);

cvNamedWindow("oo_erode",1);

input = cvLoadImage("apple.jpg",1);

cvShowImage( "ii", input );

//make erode and dilate, clones of input (remember that cloning automatically copies height, width etc.)

dilate=cvCloneImage( input );
```

```
erode=cvCloneImage( input );
//dilate image
cvDilate(input,dilate,structure_element ,1);
//cvDilate(input image pointer , output image pointer , structural element , number of iterations)
//erode image
cvErode(input,erode,NULL,1);
//cvErode(input image pointer , output image pointer , structural element , number of iterations)
cvShowImage( "oo_dilate", dilate);
cvShowImage( "input", input);
cvShowImage( "oo_erode", erode );
cvWaitKey(0);
cvDestroyWindow( "ii" );
 cvDestroyWindow( "oo_dilate" );
 cvDestroyWindow( "oo_erode" );
cvReleaseImage( &input );
cvReleaseImage( &dilate );
cvReleaseImage( &erode );
return 0;
}
```

4. **Thresholding an image**

Thresholding an image is one of the simplest ways of image segmentation.

As the name suggests, it carries out its change according to a set threshold.

To threshold an image following function is used:

cvThreshold(input, output, threshold, maxValue, thresholdType)]

Following threshold types are available

- **CV_THRESH_BINARY**
  -----max value if more than threshold, else 0
- **CV_THRESH_BINARY_INV**
  -----0 if more than threshold, else max value
- **CV_THRESH_TRUNC**
  -----threshold if more than threshold, else no change
- **CV_THRESH_TOZERO**
  ------no change if more than threshold else 0
- **CV_THRESH_TOZERO_INV**
  ------0 if more than threshold, else no change

Before we threshold the image we need to make a clone of the image.

Example image for binary image has been obtained from the example grayscale image of 'beautiful Lena'.

5. Image data

An image's data is stored as a character array whose first element is pointed by:-

Input->imageData (char pointer)

Number of array elements in 1 row is stored in

input->widthStep

Accessing pixel values in a grayscale image:

To find the pixel value in an image we need to define a pointer of type **uchar:**

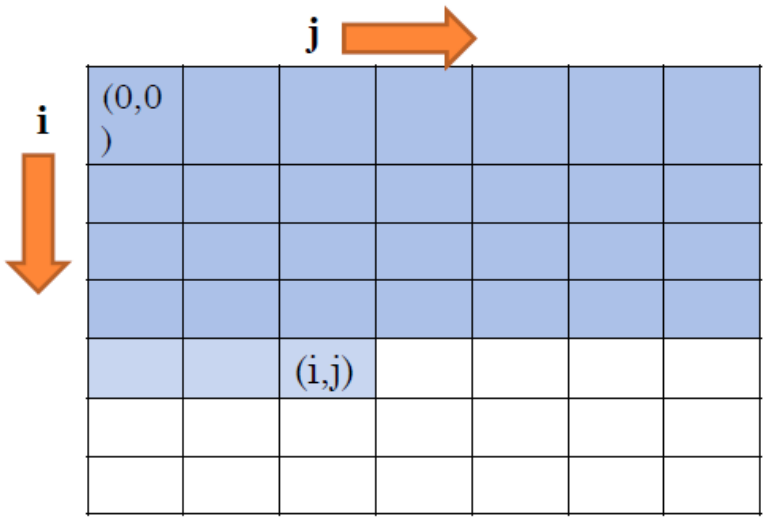
uchar *pinput = (uchar*)input->imageData;

Following image will explain how an image is accessed.

To get the (i,j) pixel value we start traversing the first row pixel by pixel

Until we reach the end of it and then move to the next row and

continue the process until we reach the (i,j) pixel.



int c = pinput[i*input->widthStep + j];

--------stores the pixel value of (i,j) pixel in c

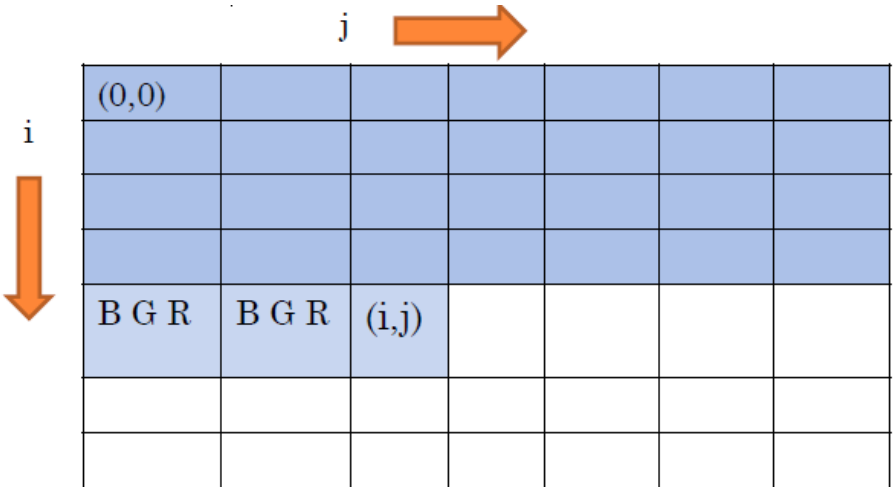
For a RGB image

uchar *pinput = (uchar*)input->imageData;

int b= pinput[i*input->widthStep + j*input->nChannels+0];

int g= pinput[i*input->widthStep + j*input->nChannels+1];

int r= pinput[i*input->widthStep + j*input->nChannels+2];



Let's play with image data…. The following code will turn a RGB image into an image which has just red objects and other values assigned to

zero and prints all image related data.

```
#include "cv.h"
#include "highgui.h"
int main()
{
   int i , j;
   IplImage* input;
   IplImage* output;
   input=cvLoadImage(“input.jpg",1);
   cvNamedWindow("ii",1);
   cvShowImage("ii",input);
   printf("nChannels=%d width=%d  height=%d widthstep=%d  depth=%d
        align=%d",input->nChannels,input->width,input->height,input-
        >widthStep,input->depth,input->align);
  output=cvCreateImage(cvSize(input->width, input->height ),input-
        >depth, input->nChannels );
   uchar *pinput   =  (uchar*)input->imageData;
//saving data pointer of input image as pinput
   uchar *poutput   = ( uchar* )output->imageData;
//saving data pointer of output image as poutput
   for(i=0;i<input->height;i++)
         for(j=0;j<input->width;j++)
```

```
        {
            poutput[i*input->widthStep + j*input->nChannels + 2]
                =pinput[i*input->widthStep + j*input->nChannels + 2];
//copying red elements of input to output
                poutput[i*input->widthStep + j*input->nChannels + 0]=0;
//initialising blue elements of output image as 0
                poutput[i*input->widthStep + j*input->nChannels + 1]=0;
//initializing green elements of output image as 0;
 //Note: initialing B and G as 0 may be excluded but recommended as it
        may take garbage value, test it yourself
        }
  cvNamedWindow("aa",1);
   cvShowImage("aa",output);
   cvWaitKey(0);
   cvDestroyWindow("ii");
   cvDestroyWindow( "aa" );
   cvReleaseImage( &output );
    cvReleaseImage( &input );
    return 0;
}
```

6. Video Input

What is a video? A video is basically a collection of continuous images displayed at a certain rate (generally 30 frames per second).

To extract the frames from video first we need to attach this video to the input stream and then extract those as and when required.

To attach the video to input stream we use the following function

CvCapture* capture=cvCreateFileCapture("file_name.avi");

And for extracting frames use the following function:

Ipl Image* input_frame=cvQueryFrame(capture);

7. Camera Input

First camera needs to be initialized and then image is captured and further operations can be carried out on that image.

Use the following command for initiating the camera:

CvCapture *capture=cvCreateCameraCapture(0);

0 stands for default webcam and to access other camera connected to the computer use 1 as argument.

Starting a camera takes time, so make sure that sufficient time has passed before we capture the image. This can be achieved through

for(int i=0;i<100000000&& capture!=NULL ;i++);

Finally image is captured and stored in variable of type IplImage*

frame=cvQueryFrame(capture);

8. Video input through camera

This is similar to video input all you need is attach the video from camera to the input stream. Following function helps us do so:

CvCapture *capture=cvCreateCameraCapture(0);

There is no need to initialize the camera in this case because frame is captured regularly. Again, 0 for default webcam and use 1 for input through external camera.

9. Playing with the mouse

For moving the mouse we use the following function declaration:

```
void Mouse_Move(DWORD dx,DWORD dy)
{
    DWORD event=0;
    event = MOUSEEVENTF_ABSOLUTE|MOUSEEVENTF_MOVE;
    mouse_event(event, dx*65535/Get_ScreenWidth(),
dy*65535/Get_ScreenHieght(), 0, 0);
}
```

Function definition for Get_ScreenWidth():

```
LONG Get_ScreenWidth()
{
    RECT rect;
    GetWindowRect(GetDesktopWindow(),&rect); //Get Desktop rect
    return rect.right - rect.left;
```

```
}
```

Function definition for Get_Screen_Hieght():

```
LONG Get_ScreenHieght()
{
        RECT rect;
        GetWindowRect(GetDesktopWindow(),&rect); //Get Desktop rect
        return rect.bottom - rect.top;
}
```

Pass the x and y co-ordinates of the screen as parameters and mouse pointer will be moved to the corresponding location.

Following function prints the RGB values of the pixel at which mouse pointer is pointing to.

```
void my_mouse_callback( int event, int x, int y, int flags, void* param )
{
  uchar *pimage   =  (uchar*)image->imageData;
printf("\nx=%d\t y=%d\n r=%d \tg=%d \tb=%d\n",x,y,
pimage[y*image->widthStep + x*image->nChannels+2], pimage[y*image->widthStep + x*image->nChannels+1], pimage[y*image->widthStep + x*image->nChannels+0]);
}
```

To call the above declared function use the following:

```
cvNamedWindow("image",1);
 cvSetMouseCallback("image", my_mouse_callback, NULL);
  cvShowImage("image",image);
```

10. Displaying an Image in Full screen

Displaying an image in full screen basically means getting rid of the borders. This can be done by using a handle for the image.

```
//generate window
cvNamedWindow("main_win", CV_WINDOW_AUTOSIZE);
//move it to initial
cvMoveWindow("main_win", 0, 0);
//set it's size to maximum possible
cvSetWindowProperty("main_win", CV_WINDOW_FULLSCREEN, CV_WINDOW_FULLSCREEN);
//show the image
cvShowImage("main_win", cv_img);
//set up the handle for the image
HWND win_handle = FindWindow(0, "main_win");
//if handle fails to load
if (!win_handle)
{
  printf("Failed FindWindow\n");
}
```

```
//modify the 'handle' so that 'window' is deprived of borders

SetWindowLong(win_handle, GWL_STYLE, GetWindowLong(win_handle,
GWL_EXSTYLE) | WS_EX_TOPMOST);

//show the new window

ShowWindow(win_handle, SW_SHOW);
```

Now, some playing with human features:

11. Haar Cascades

Haar like features are digital image features used in object recognition. Haar Cascades are trained classifiers used for detecting features like face, eyes, upper body etc.

These cascades are stored in the data folder of OpenCV.

Firstly, you need to load cascade and then use the cascade to detect the presence of the corresponding feature. In most cases you need to mark the region of your interest. Following code detects eyes and marks a rectangle around the eyes.

```
#include "cv.h"

#include "highgui.h"

#include "math.h"

#include "cxcore.h"

static CvMemStorage* storage = 0;

static CvHaarClassifierCascade* cascade = 0;

const char* cascade_name =
"C:/OpenCV2.1/data/haarcascades/haarcascade_eye.xml";

// This is the address of the cascade used for eye detection on my machine
```

```
void detect_and_draw( IplImage* img );
int main()
{
   IplImage* frame;    //Initialise input image pointer
   cascade = (CvHaarClassifierCascade*)cvLoad( cascade_name, 0, 0, 0 );
   int c;
    if( !cascade )
    {
        fprintf( stderr, "ERROR: Could not load classifier cascade\n" );
        return -1;
    }
   frame = cvLoadImage("reformed.jpg",1);
   storage = cvCreateMemStorage(0);
   detect_and_draw(frame);
   cvWaitKey(0);
  return 0;
}

void detect_and_draw( IplImage* img )
 {
    int scale = 1;
```

```
    // Create a new image based on the input image
    IplImage* temp = cvCreateImage( cvSize(img->width/scale,img->height/scale), 8, 3 );
    // Create two points to represent the face locations
    CvPoint pt1, pt2;
    int i;
    // Clear the memory storage which was used before
    cvClearMemStorage( storage );
    // Find whether the cascade is loaded, to find the faces. If yes, then:
if( cascade )
    {
     // There can be more than one face in an image. So create a growable sequence of faces.
     // Detect the objects and store them in the sequence
      CvSeq* faces = cvHaarDetectObjects( img, cascade, storage,
                                          1.1, 2, CV_HAAR_DO_CANNY_PRUNING,
                                          cvSize(40, 40) );
      // Loop the number of faces found.
      for( i = 0; i < (faces ? faces->total : 0); i++ )
      {
        // Create a new rectangle for drawing the face
        CvRect* r = (CvRect*)cvGetSeqElem( faces, i );
```

```
            // Find the dimensions of the face, and scale it if necessary
            pt1.x = r->x*scale;
            pt2.x = (r->x+r->width)*scale;
            pt1.y = r->y*scale;
            pt2.y = (r->y+r->height)*scale;
            // Draw the rectangle in the input image
            cvRectangle( img, pt1, pt2, CV_RGB(255,0,0), 3, 8, 0 );
        }
    }
    // Show the image in the window named "result"
   cvShowImage( "result", img );
    // Release the temp image created.
    cvReleaseImage( &temp );
}
```

12. <u>Cropping an Image</u>

```
cvSetImageROI(img, cvRect(origin_x,origin_y, width, hieght));
IplImage *face = cvCreateImage(cvGetSize(img),img->depth,img->nChannels);
 cvCopy(img, face, NULL);        //Copies interested area of image in face
 cvResetImageROI(img);
```

13. Blob detection edge detection

These are some of other actions which can be performed using OpenCV

And there are numerous other interesting features of OpenCV available for playing with the images. Enjoy! Playing with images