Min Sun

# OpenCV
# +
# Face Detection

**COS429**
Computer Vision

# Today's outline

- The OpenCV Library
  - Brief introduction
  - Getting started
- Creating a face detector
  - How it's done
  - OpenCV implementation
- Using a face detector
  - Example code, step-by-step

# The OpenCV Library – brief introduction

- Open source computer vision library written in C/C++, created and maintained by Intel
- Cross-platform and extremely portable
- Targeted for real-time applications

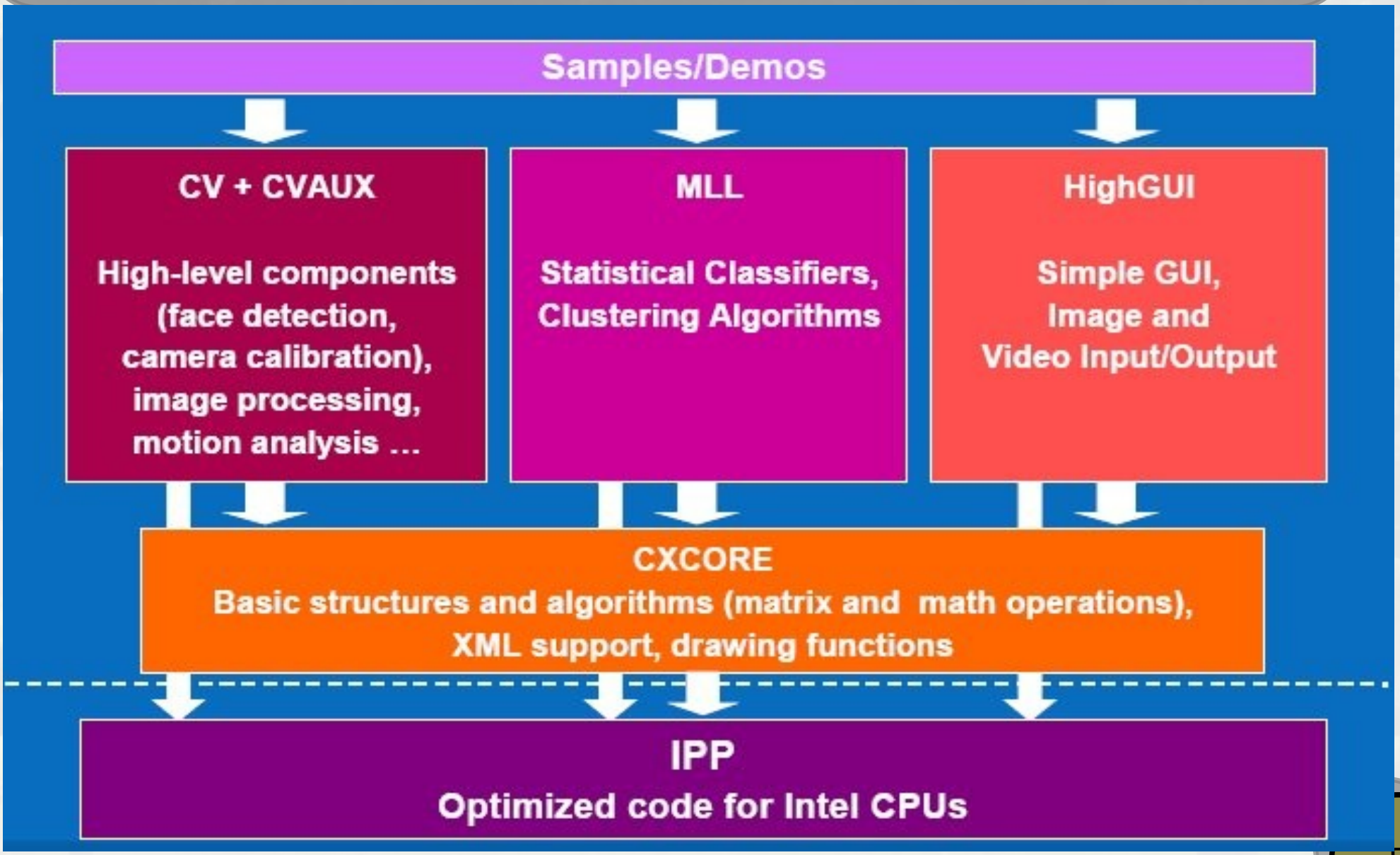
**Supported Platforms**

| | IA32 (x86) | EM64T (x64) | IA64 (Itanium) | Other (PPC, Sparc) |
|---|---|---|---|---|
| Windows | ✓ (w. IPP; VS98,VS2005(OpenMP),ICC,GCC, BCC) | ✓(w. IPP, VS98+PSDK, VS2005(OpenMP)) | Partial Support | N/A |
| Linux | ✓ (w. IPP; GCC, ICC) | ✓ (w. IPP; GCC, ICC) | ✓ (GCC, ICC) | ✗ |
| MacOSX | ✓ (w. IPP, GCC, native APIs) | ? (not tested) | N/A | ✓ (iMac G5, GCC, native APIs) |
| Others (BSD, Solaris...) | ✗ | ✗ | ✗ | Reported to build on UltraSparc, Solaris |

# The OpenCV Library – getting started

- Download OpenCV:
  - http://sourceforge.net/projects/opencvlibrary/
- Setting up
  - There's a comprehensive guide on setting up OpenCV in various environments at the official wiki: http://opencvlibrary.sourceforge.net/

# Library Architecture



# Modules Descriptions: HighGUI

## Functionality Overview

“Smart” windows
Image I/O, rendering
Processing keyboard and other events, timeouts
Trackbars
Mouse callbacks
Video I/O

# Windows

cvNamedWindow(window_name, fixed_size_flag);

creates window accessed by its name. Window handles repaint, resize events. Its position is remembered in registry:

```
cvNamedWindow("ViewA",1);
cvMoveWindow("ViewA",300,100);
cvDestroyWindow("ViewA");
...
```

cvShowImage(window_name, image);

copies the image to window buffer, then repaints it when necessary. {8u|16s|32s|32f}{C1|3|4} are supported.

only the whole window contents can be modified. Dynamic updates of parts of the window are done using operations on images, drawing functions etc.

On Windows native Win32 UI API is used

On Linux – GTK+ 2.x

On MacOSX – Carbon.

# I/O

IplImage* cvLoadImage(filename, colorness_flag);

loads image from file, converts it to color or grayscale, if needed.

image format is determined by the file contents.

cvSaveImage(filename, image);

saves image to file, image format is defined by filename extension.

Supported formats: BMP, JPEG (libjpeg), JPEG 2000 (Jasper), PNG (libpng), TIFF (libtiff), PPM/PGM.

```
// converting jpeg to png
IplImage* img = cvLoadImage("picture.jpeg",-1);
if( img ) cvSaveImage( "picture.png", img );
```

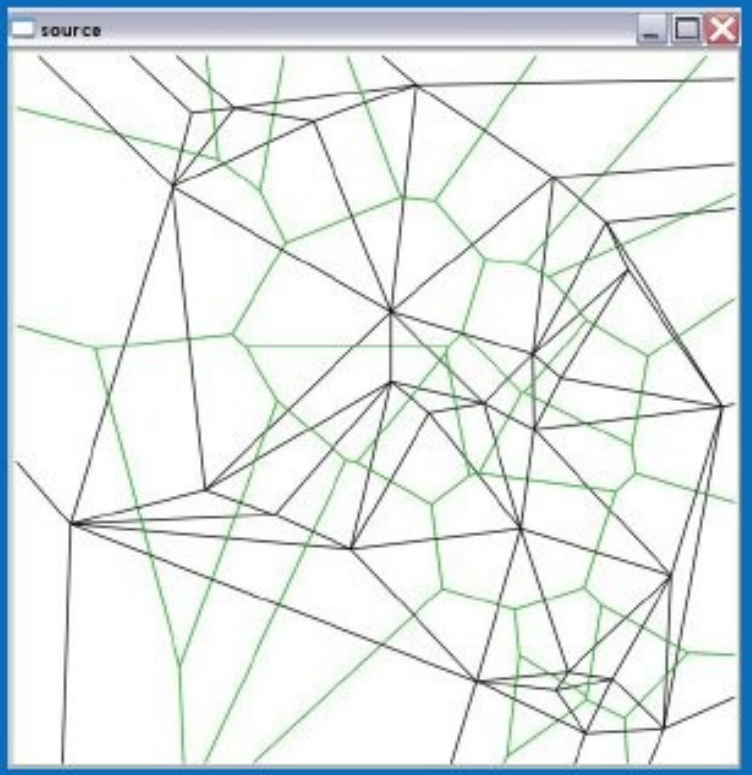
# Waiting for keys...

cvWaitKey(delay=0);

waits for key press event for <delay> ms or infinitely, if delay is 0.

To make program continue execution if no user actions is taken, use cvWaitKey(<delay_ms!=0>); and check the return value

```
// opencv/samples/c/delaunay.c
...
for(...) {
    ...
    int c = cvWaitKey(100);
    if( c >= 0 )
        // key_pressed
        break;
}
```



# The OpenCV Library – getting started

## Hello World!

```
//This code is taken from http://www.cs.iit.edu/~agam/cs512/lect-notes/opencv-intro/opencv-
//intro.html#SECTION00025000000000000000

//////////////////////////////////////////////////////////////////////////
//
// hello-world.cpp
//
// This is a simple, introductory OpenCV program. The program reads an
// image from a file, inverts it, and displays the result.
//
//////////////////////////////////////////////////////////////////////////
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <cv.h>
#include <highgui.h>

int main(int argc, char *argv[])
{
  IplImage* img = 0;
  int height,width,step,channels;
  uchar *data;
  int i,j,k;

  if(argc<2){
    printf("Usage: main <image-file-name>\n\7");
    exit(0);
  }

  // load an image
  img=cvLoadImage(argv[1]);
  if(!img){
    printf("Could not load image file: %s\n",argv[1]);
    exit(0);
  }
```

# The OpenCV Library – getting started

## Hello World! (cont'd)

```
  // get the image data
  height    = img->height;
  width     = img->width;
  step      = img->widthStep;
  channels  = img->nChannels;
  data      = (uchar *)img->imageData;
  printf("Processing a %dx%d image with %d channels\n",height,width,channels);

  // create a window
  cvNamedWindow("mainWin", CV_WINDOW_AUTOSIZE);
  cvMoveWindow("mainWin", 100, 100);

  // invert the image
  for(i=0;i<height;i++) for(j=0;j<width;j++) for(k=0;k<channels;k++)
    data[i*step+j*channels+k]=255-data[i*step+j*channels+k];

  // show the image
  cvShowImage("mainWin", img );

  // wait for a key
  cvWaitKey(0);

  // release the image
  cvReleaseImage(&img );
  return 0;
}
```

# Creating a face detector

- Original Viola-Jones paper (CVPR, 2001)
  - http://research.microsoft.com/~viola/Pubs/Detect/violaJones_CVPR2001.pd
- Collection of training images, positives and negatives
- Run AdaBoost to distill a set of Haar-like features which give good classifiers
- Combine the yielded classifiers appropriately into a cascade

# Creating a face detector – cont'd

- Good news – OpenCV comes with an implementation of Viola-Jones!
- A good reference - http://note.sonots.com/SciSoftware/haartraining.html
- Three tools to use – “createsamples”, “haartraining” and “performance”
- createsamples
  - Tool from OpenCV to automate creation of training samples
  - Four functionalities
    - 1. create training samples from one image applying distortions
    - 2. create training samples from from a collection of images, without distortions
    - 3. create testing samples with ground truth from one image applying distortions
    - 4. show images from the .vec internal format which contains a collection of samples
    - Best to use a combination of the functionalities to create many samples from many images with distortions and merge them

# Creating a face detector – cont'd

- haartraining
  - The software that performs the viola-jones algorithm and creates the cascade file
  - Sample run:
    ```
    opencv-haartraining -data class -vec samples.vec -bg negs.txt
    -nstages 20 -nsplits 2 -minhitrate 0.999 -maxfalsealarm 0.5 -npos
    3420 -nneg 4800 -w 24 -h 24 -mem 3072 -mode BASIC
    ```
    - “data” is the directory in which to store the output
    - “vec” is the .vec file containing the positive images
    - “bg” is a text file with a collection of paths to background (negative) images
    - “nstages” is the number of stages of boosting
    - “nsplits” is the number of split nodes in the decision trees which are the weak classifier for boosting
    - “minhitrate” and “maxfalsealarm” are cutoff values for hit rate and false alarm, per stage
    - “npos” and “nneg” are the number of positive and negative images to be used from the given sets
    - “w” and “h” are the width and the height of the sample
    - “mem” is the amount of memory that should be used for precalculation. The default is 200MB
    - “mode” is either “BASIC” or “ALL”, stating whether (ALL) the full set of Haar features should be used (both upright and 45 degree rotated) or (BASIC) only upright features

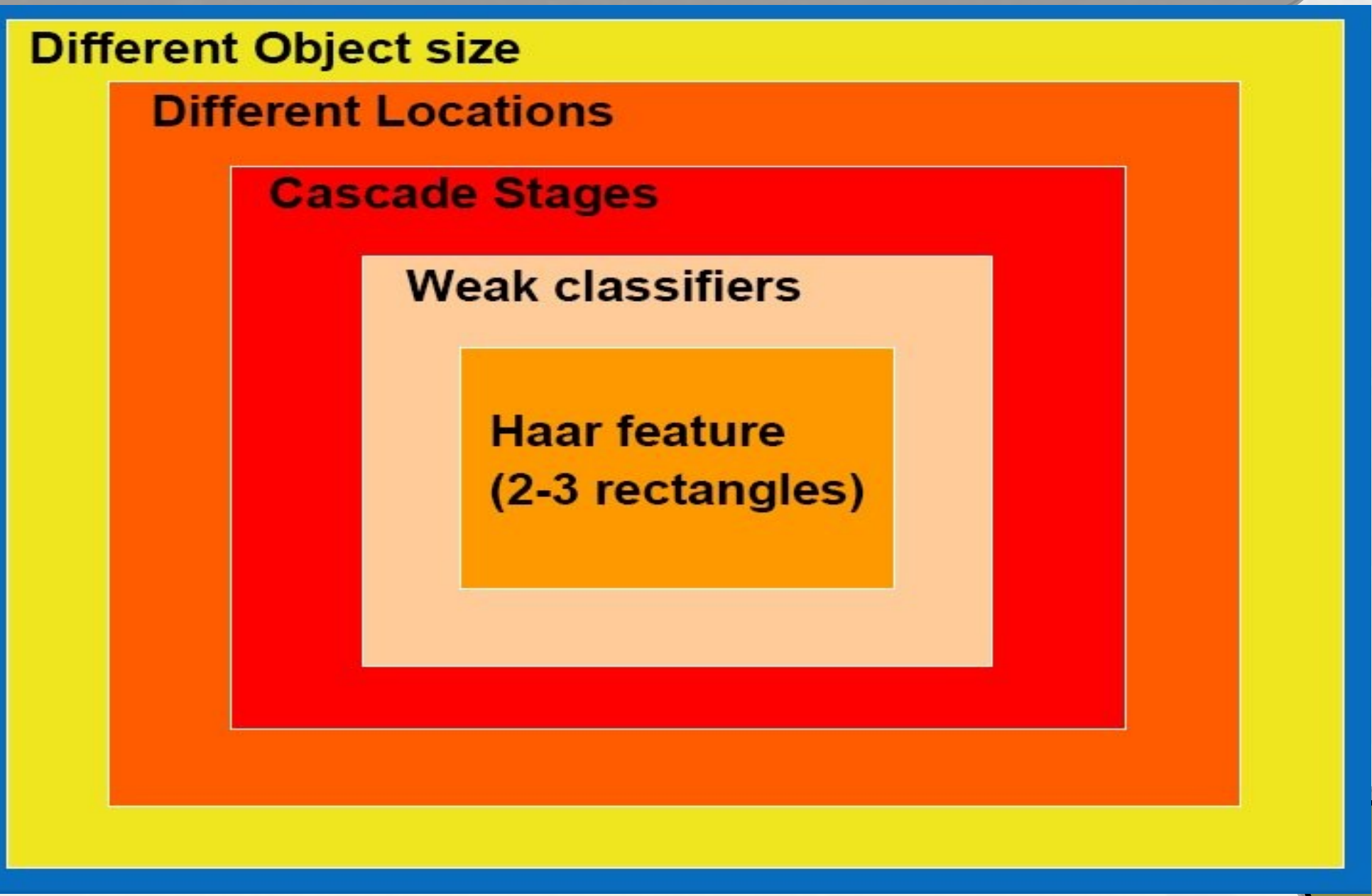
# Creating a face detector – cont'd

- performance
  - A tool to test the performance of the obtained face detector, given the testing set of annotated positives and negatives (created with createsamples)
  - Input is the haartraining output dir, and the description file for testing samples generated by createsamples
  - Output is the numbers of correctly detected objects, not detected objects, and false positives (detected objects which do not exist)

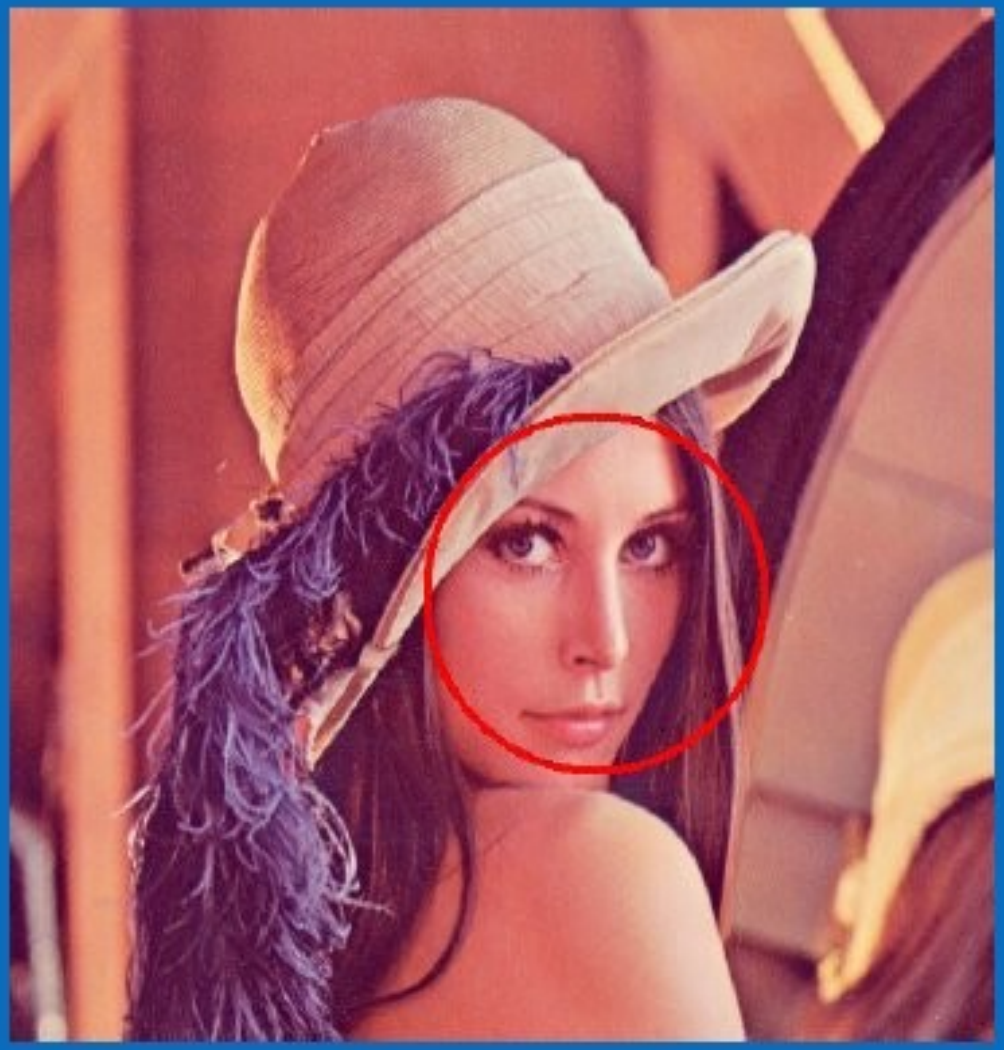
# Creating a face detector – cont'd

- Good news – OpenCV also comes with several cascade files for detecting both frontal and profile faces
- Bad news – These work with “real” photographs, won't work well for the cartoony frames in your final project
- Good news – you just learned how to train your own cascade classifier
- Bad news – it will take days on modern computers (even with multiple processors)
- Good news – you get a face detector for wii avatars from us!

# Detector Algorithm



# Non maxima suppression



# Using a face detector (code)

```
#include "cv.h"
#include "highgui.h"
#include <stdlib.h>

#ifdef _EiC
#define WIN32
#endif

const char* cascade_name = "wii_frontalface4.xml";

int main ( int argc, const char* argv[] )
{
    /* Load the classifier data from the .xml file */
    CvHaarClassifierCascade* cascade = (CvHaarClassifierCascade*)cvLoad( cascade_name);

    /* create a window with handle result */
    cvNamedWindow( "result" );

    /* read the input image */
    IplImage* image = cvLoadImage( argv[1], 1 );

    /*
       create a b&w image of the same size object as a placeholder for now
       - cvSize(width, height) is the object representing dimensions of the image
       - 8 stands for 8-bit depth
       - 1 means one-channel (b&w)
       Hence a 24bit RGB image of size 800x600 would be created with
         cvCreateImage( cvSize(800, 600), 8, 3);
    */
    IplImage* gray = cvCreateImage( cvSize(image->width,image->height), 8, 1 );

    /* now convert the input image into b&w and store it in the placeholder */
    cvCvtColor( image, gray, CV_BGR2GRAY );

    /* create memory storage to be used by cvHaarDetectObjects */
    CvMemStorage* storage = cvCreateMemStorage();
```

# Using a face detector (code) cont'd

```
/* used cvHaarDetectObjects */

/* 8-bit depth RGB representation of color red */
static CvScalar RED = {0, 0, 255};

/* go through all the detected faces, and draw them into the input image */
for (int i = 0; i < (faces ? faces->total : 0); i++)
{
    CvRect *r = (CvRect*)cvGetSeqElem( faces, i );
    CvPoint ul; CvPoint lr;
    ul.x = r->x;              ul.y = r->y;
    lr.x = r->x + r->width;   lr.y = r->y + r->height;

    /* draws a rectangle with given coordinates of the upper left
       and lower right corners into an image */
    cvRectangle(image, ul, lr, RED, 3, 8, 0);
}
```

# Using a face detector (code) cont'd

```
    /* free up the memory */
    cvReleaseImage( &gray );

    /* show the result and wait for a keystroke form user before finishing */
    cvShowImage( "result", image );
    cvWaitKey(0);
    cvReleaseImage( &image );
    cvDestroyWindow("result");

    return 0;
}
```

# Using a face detector (code) cont'd

result: