Introduction to OpenCV

Vadim Pisarevsky
Senior Software Engineer
Intel Corporation, Software and Solutions Group

Agenda

General Information

Getting Started with OpenCV

Modules Description

Interaction with Intel IPP

Python Interface

Some Usage Examples

Summary

General Information

What is OpenCV?

OpenCV stands for Open Source Computer Vision Library

Includes over **500** functions implementing computer vision, image processing and general-purpose numeric algorithms.

Portable and very efficient (implemented in C/C++)

Has BSD-like license (that is, absolutely free for academic and commercial use)

Available at http://sourceforge.net/projects/opencvlibrary

Why OpenCV?

Computer Vision Market is large and continues to grow

There is no standard API (like OpenGL and DirectX in graphics, or OpenSSL in cryptography), most of CV software is of 3 kinds:

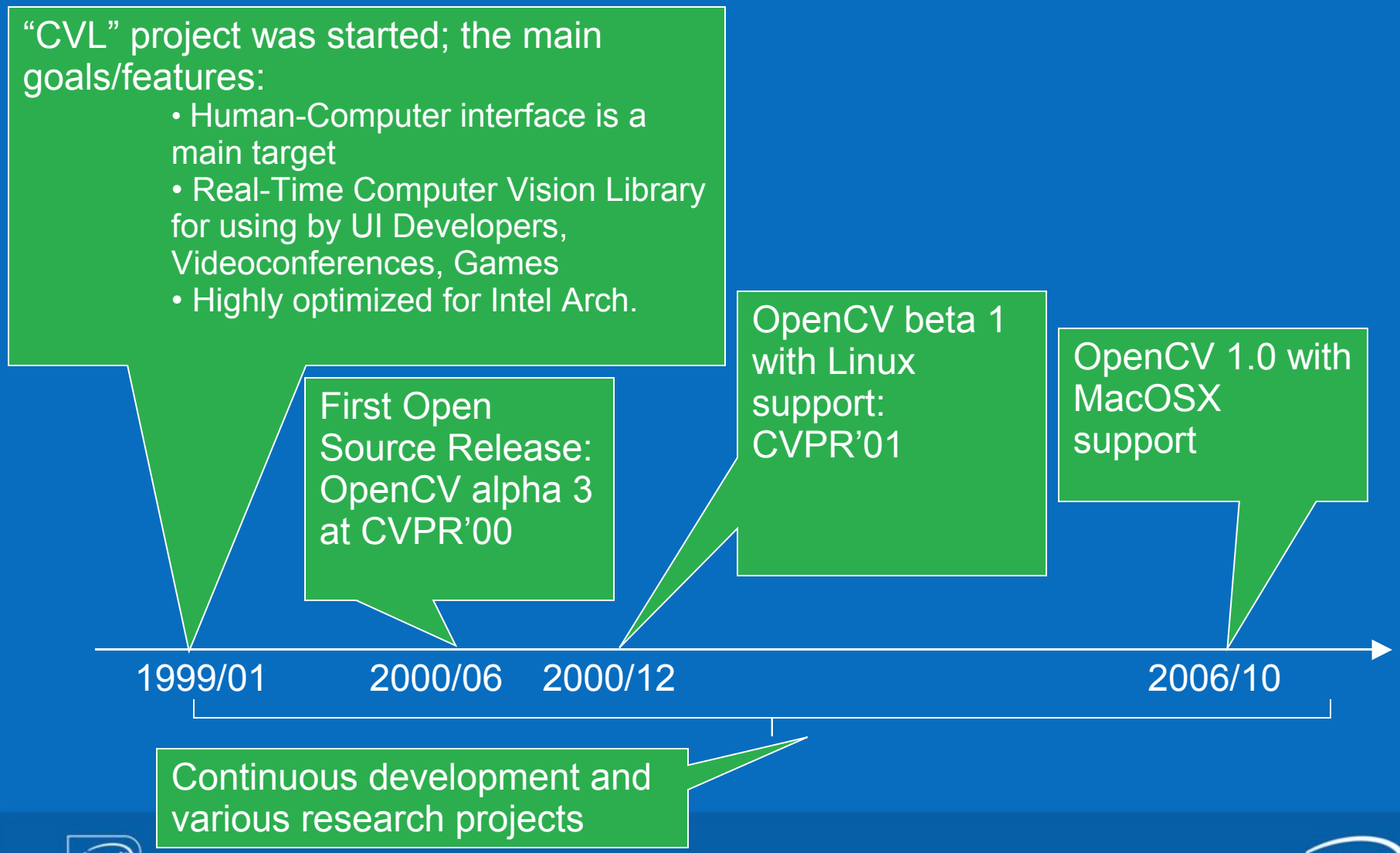
- Research code (slow, unstable, independent/incompatible data types for every library/toolbox)
- Very expensive commercial toolkits (like Halcon, MATLAB+Simulink, …)
- Specialized solutions bundled with hardware (Video surveillance, Manufacturing control systems, Medical equipment …)

Standard library would simplify development of new applications and solutions much easier.

Special optimization for Intel Architectures:

- Creates new usage models by achieving real-time performance for quite "heavy" algorithms (like face detection)
- Makes Intel platforms attractive for CV developers

The Project History



OpenCV Community

The library is actively used by a large number of companies (such as Intel, IBM, Microsoft, SONY, Siemens, Google,…) and research centers (Stanford, MIT, CMU, Cambridge, INRIA etc.)

>14000 members of the forum OpenCV@yahoogroups.com, with average daily traffic ~10-20 messages.

Community contributes to the project: bug reports, patches, new features (video acquisition, 3d tracking, textures, Python interface)

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

Library Design

Initial goal: build high-level ready-to-use components, like gesture recognition, camera calibration etc.

But … we have fundamental Computer Vision problem: can not create CV system that works always and everywhere (because of diff. lighting conditions, subject variety …) => need to analyze each problem.

"Analysis": *1581, "resolution of anything complex into simple elements"*

We split a complex problem into elements/building blocks. Each block may require further decomposition. Then for every block (compound or primitive) we either use existing functions or create our own => we finally got multi-level library, which components may be re-used

Example: Gesture Recognition

How to find hand?
- By color
- Using motion detection
- Depth information (w. stereo camera)

…

How to extract the shape?

What are the lighting conditions?

Dynamic or static gestures?

Pre-defined set of gestures

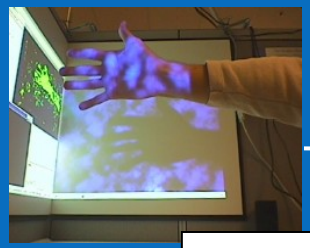
or extendable by user?

So for this algorithm we need stereo,
3D clustering, color histograms,
image moments,
statistical classification
(or just Mahalanobis distance)
and the way to visualize all the algorithm stages for debugging

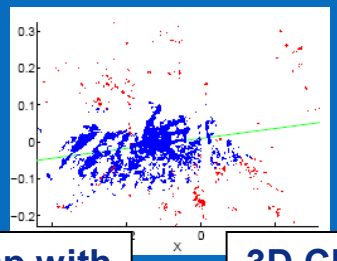
no free libraries could do all/most of this! => We had to do it ourselves

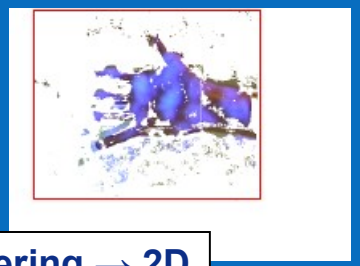
One of algorithms:

?



Depth Map with Stereo Camera





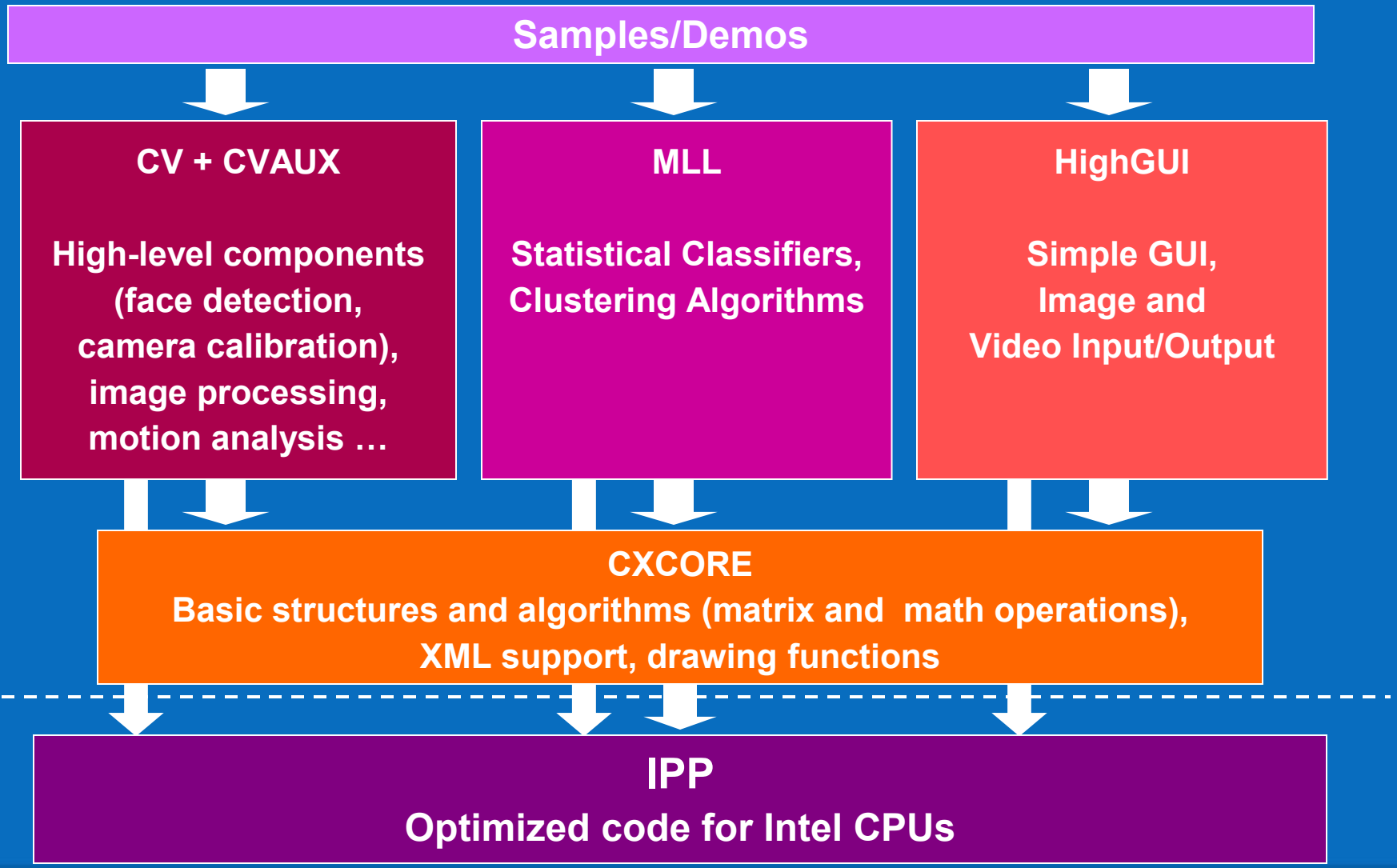
3D Clustering → 2D

Color Histograms

+

Statistical Classification (Image moments as features)

Library Architecture

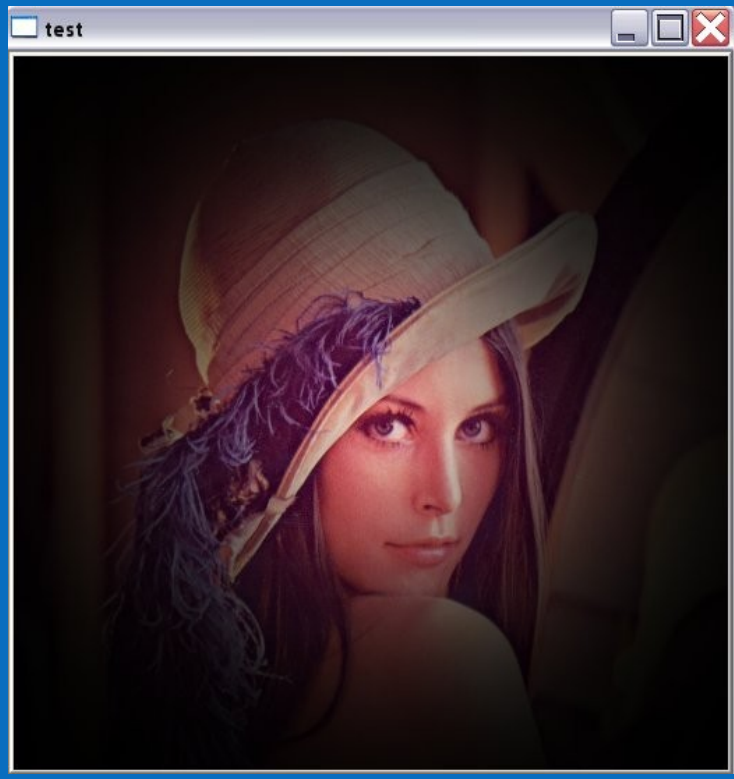


Getting Started with OpenCV

The First OpenCV Program


```
1. #include <cv.h>
2. #include <highgui.h>
3. #include <math.h>
4. int main( int argc, char** argv ) {
5.     CvPoint center;
6.     double scale=-3;
7.     IplImage* image = argc==2 ? cvLoadImage(argv[1]) : 0;
8.     if(!image) return -1;
9.     center = cvPoint(image->width/2,image->height/2);
10.    for(int i=0;i<image->height;i++)
11.        for(int j=0;j<image->width;j++) {
12.            double dx=(double)(j-center.x)/center.x;
13.            double dy=(double)(i-center.y)/center.y;
14.            double weight=exp((dx*dx+dy*dy)*scale);
15.            uchar* ptr =
    &CV_IMAGE_ELEM(image,uchar,i,j*3);
16.            ptr[0] = cvRound(ptr[0]*weight);
17.            ptr[1] = cvRound(ptr[1]*weight);
18.            ptr[2] = cvRound(ptr[2]*weight);  }
19.    cvSaveImage( “copy.png”, image );
20.    cvNamedWindow( "test", 1 );
21.    cvShowImage( "test", image );
22.    cvWaitKey();
23.    return 0; }
```

Radial gradient



How to build and run the program?

Obtain the latest version of OpenCV:

a) Get the stable release (currently, 1.0) at http://www.sourceforge.net/projects/opencvlibrary

b) Or get the latest snapshot from CVS:
:pserver:anonymous@opencvlibrary.cvs.sourceforge.net:/cvsroot/opencvlibrary

Install it:

Windows: run installer, open opencv.sln (opencv.dsw) with Visual Studio and build the solution

Linux/MacOSX: use ./configure + make + make install (see the INSTALL document)

Build the sample:

Windows:

- cl /I<opencv_inc> sample.cpp /link /libpath:<opencv_lib_path> cxcore.lib cv.lib highgui.lib
- Create project for VS (see opencv/docs/faq.htm)
- or use opencv/samples/c/cvsample.vcproj (cvsample.dsp) as starting point

Linux:

```
g++ -o sample `pkg-config –cflags` sample.cpp `pkg-config –libs opencv`
```

Run it: sample lena.jpg

The Program Analysis Line-by-Line

```
1. #include <cv.h>
2. #include <highgui.h>
3. #include <math.h>
4. int main( int argc, char** argv ) {
5.     CvPoint center;
6.     double scale=-3;
7.     IplImage* image = argc==2 ? cvLoadImage(argv[1]) : 0;
8.     if(!image) return -1;
9.     center = cvPoint(image->width/2,image->height/2);
10.    for(int i=0;i<image->height;i++)
11.        for(int j=0;j<image->width;j++) {
12.            double dx=(double)(j-center.x)/center.x;
13.            double dy=(double)(i-center.y)/center.y;
14.            double weight=exp((dx*dx+dy*dy)*scale);
15.            uchar* ptr =
       &CV_IMAGE_ELEM(image,uchar,i,j*3);
16.            ptr[0] = cvRound(ptr[0]*weight);
17.            ptr[1] = cvRound(ptr[1]*weight);
18.            ptr[2] = cvRound(ptr[2]*weight);  }
19.    cvSaveImage( “copy.png”, image );
20.    cvNamedWindow( "test", 1 );
21.    cvShowImage( "test", image );
22.    cvWaitKey();
23.    return 0; }
```

1-23: The code is clear, short and cross-platform, no need to use with MFC/GTK/QT/…, cross-platform

1-2. Include OpenCV headers first

7. IplImage is OpenCV image type

7, 19. Use cvLoadImage and cvSaveImage to load/save an image in multiple formats: JPEG, JPEG2000, BMP, PNG, TIFF, PPM.

9. Use cvPoint and other “constructor” functions to initialize simple structures on fly.

15. Use CV_IMAGE_ELEM macro to access image pixels

16-18. Use cvRound for very fast float->int conversion.

20. Use cvNamedWindow to create self-updating highgui window

21. Use cvShowImage to display image in window

22. Use cvWaitKey to get input from user

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.

Image 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 );
```

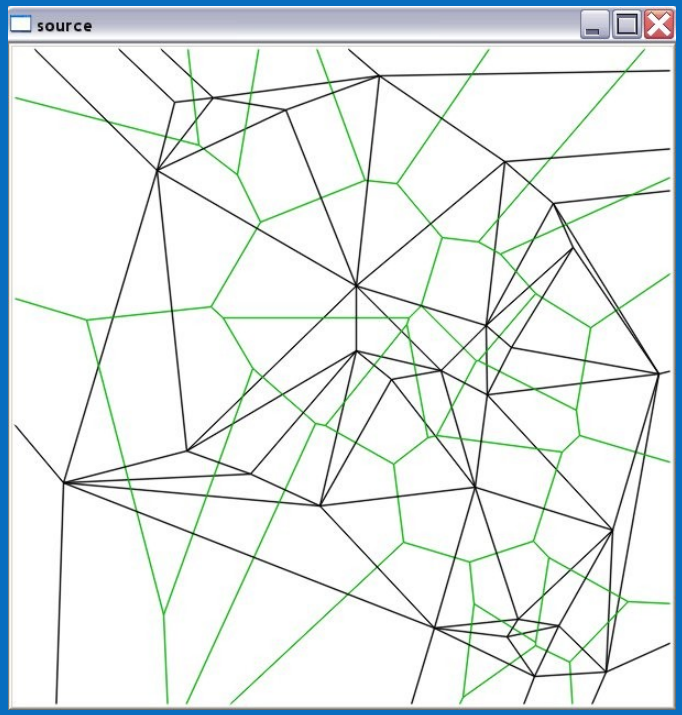
Waiting for keys...

cvWaitKey(delay=0);

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

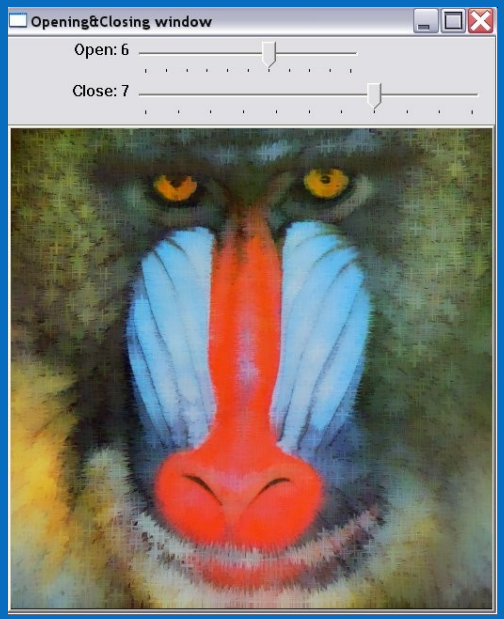
To make program continue execution if no user actions is taken, use cvWaitKey(<delay_ms!=0>); and check the return value

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



Trackbars & Mouse Callbacks

```
// opencv/samples/c/morphology.c
int dilate_pos=0; // initial position value
void Dilate(int pos) {
   … cvShowImage( “E&D”, dilate_result );
}
int main(){
   cvNamedWindow(“E&D”,1);
   cvCreateTrackbar(“Dilate”,”E&D”, &dilate_pos,10,Dilate);
   …
   cvWaitKey(0); // check for events & process them
}
```




```
// opencv/samples/c/lkdemo.c
void on_mouse(int event,int x,int y,int flags,  void* param) { … }
int main(){
cvNamedWindow(“LkDemo”,1);
cvSetMouseCallback(“LkDemo”,on_mouse,0);
…
cvWaitKey(0); // check for events & process them
}
```

Video I/O API

CvCapture* cvCaptureFromCAM(camera_id=0);

initializes capturing from the specified camera

CvCapture* cvCaptureFromFile(videofile_path);

initializes capturing from the video file.

IplImage* cvQueryFrame(capture);

retrieves the next video frame (do not alter the result!), or NULL if there is no more frames or an error occured.

cvGetCaptureProperty(capture, property_id);

cvSetCaptureProperty(capture, property_id, value);

retrieves/sets some capturing properties (camera resolution, position within video file etc.)

cvReleaseCapture(&capture);

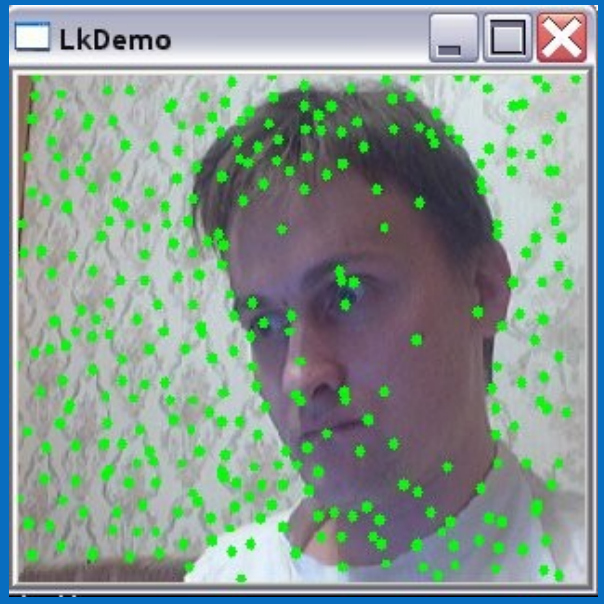
do not forget to release the resouces at the end!

Used interfaces:

- Windows: VFW, IEEE1394, MIL, DirectShow (in progress), Quicktime (in progress)
- Linux: V4L2, IEEE1394, FFMPEG
- MacOSX: FFMPEG, Quicktime

Video I/O Sample


```
// opencv/samples/c/lkdemo.c
int main(…){
…
CvCapture* capture = <…> ?
     cvCaptureFromCAM(camera_id) :
     cvCaptureFromFile(path);
if( !capture ) return -1;
for(;;) {
   IplImage* frame=cvQueryFrame(capture);
   if(!frame) break;
   // … copy and process image
   cvShowImage( “LkDemo”, result );
   c=cvWaitKey(30); // run at ~20-30fps speed
   if(c >= 0) {
      // process key
 }}
cvReleaseCapture(&capture);}
```



opencv/samples/c/lkdemo.c:
(needs camera to run)

Modules Descriptions: CXCORE

Functionality Overview

Matrix and Image Arithmetical and Logical Operations

Linear algebra, DFT/DCT, vector math

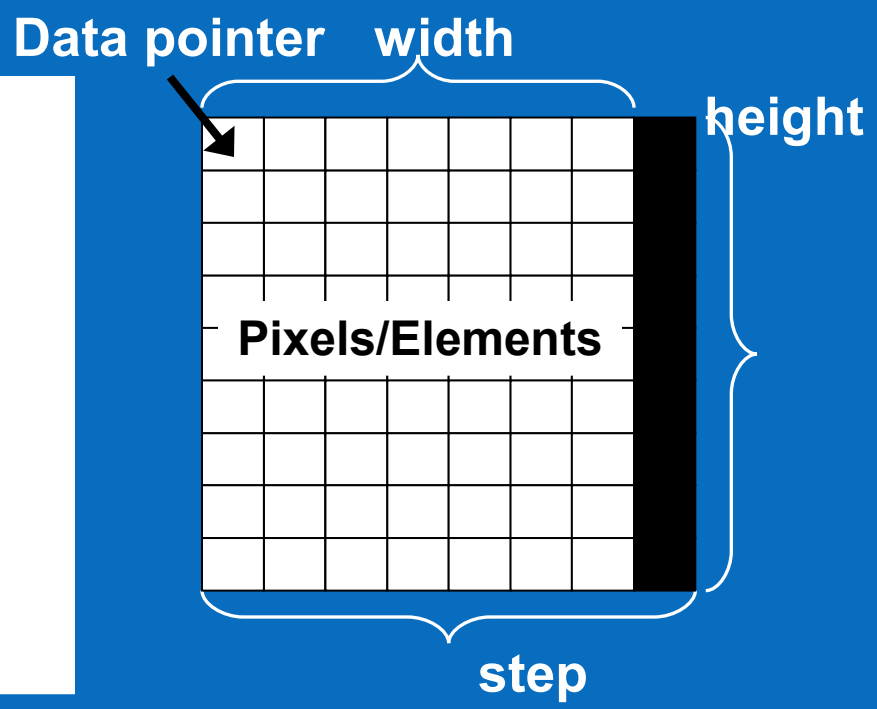
Dynamic structures

Drawing functions

XML I/O

Matrix (CvMat)

```
typedef struct CvMat {
    int type; // data type: CV_8UC1, CV_8UC3, …
              // … CV_16UC1, …, CV_64FC1
    int step;
    union { // aliased pointers to the data
       unsigned char* ptr;
       float* fl; double* db;
    } data;
    int rows; // ~width
    int cols; // ~height
    …
} CvMat;
```




```
float A[3][3] = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};
CvMat mA = cvMat( 3, 3, CV_32F, &A[0][0] ); // initialize 3x3 identity matrix
printf( "%g\n", cvDet(&mA) ); // will print 1.0
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```
// create large color image as a matrix
CvMat* image = cvCreateMat( 1024, 768, CV_8UC3 );
// fill the image with red color (blue=0, green=0, red = 255)
cvSet( image, cvScalar(0,0,255));
```

Image (IplImage)

It came from the older Intel Image Processing Library (IPL).

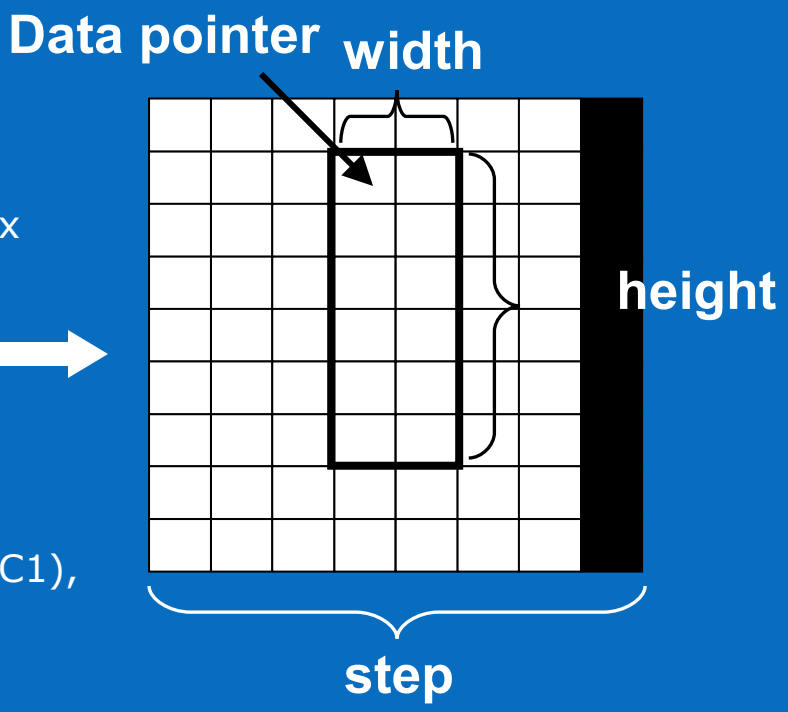
```
typedef struct _IplImage {
    int nChannels; // 1, 2, 3, 4 …
    int depth; // 8, 16, 32, 64, signed/unsigned/floating-point
    int width, height;
    int widthStep; // step
    char* imageData; // pointer to the data
    …
} IplImage;
```

```
// create color image
IplImage* image = cvCreateImage( cvSize(1024,768), 8, 3 );
// fill it with random values (color noise)
CvRNG rng = cvRNG(0x12345);
cvRandArr( &rng, image, CV_RAND_UNI, cvScalar(0,0,0),
      cvScalar(256,256,256));
```

Almost every OpenCV function that takes CvMat* accepts IplImage*, and vice versa, so you may use any (but CvMat* is preferable)

Basic operations on Matrixes/Images

```
cvReleaseMat(&A); // release matrix
CvMat* B = cvCloneMatrix(A); // make a copy of matrix
cvCopy( A, B ); // copy contents of A to B
cvSet(A, cvScalar(a,b,c,d)); cvZero(A); // initialize matrix
// Manipulating part of a matrix (no data is copied)
CvMat B; cvGetSubArr(A,&B,cvRect(3,1,2,5));
// Split matrix into a separate channels,
merge channels together
CvMat* R = cvCreateMat(RGB->rows,RGB->cols, CV_8UC1),
*G = cvCloneMat(R), *B=cvCloneMat(G);
cvSplit(RGB, B, G, R); … cvMerge(B, G, R, RGB);
```




```
// Accessing individual matrix elements
for(i=0;i<B->rows;i++) for(j=0;j<B->cols;j++) // transpose A
   CV_MAT_ELEM(*B,float,i,j)=CV_MAT_ELEM(*A,float,j,i);
```

Arithmetical and Logical Operations

```
CvMat *A, *B, *C;
CvScalar S=cvScalar(a,b,c,d);

cvAdd(A, B, C); // C = A + B
cvSub(A, B, C); // C = A – B
cvAddS(A, S, C); // C = A + S
cvSubRS(A, S, C); // C = S – A
cvMul(A, B, C, a); // C = a*A*B
cvDiv(A, B, C, a); // C = a*A/B
// C = a*A + b*B + c
cvAddWeighted( A, a, B, b, c, C);

cvCmp(A, B, C, op); // C = A op B,
                    // op is =, ≠, <, ≤, > or ≥
cvCmpS(A, a, C, op); // C = A op a
cvMin(A, B, C); // C = min(A, B)
cvMax(A, B, C); // C = max(A, B)
cvMinS(A, a, C);  // C = min(A, a);
cvMaxS(A, a, C); // C = max(A, a);
```

```
cvAnd(A, B, C); // C = A & B
cvAndS(A, a, C); // C = A & a
… cvOr, cvOrS, cvXor, cvXorS,
cvNot(A, C); // C = ~A

cvConvertScale(A,C,a,b); // C = A*a + b (A and C may
have different types!)

S = cvSum(A); // S = Σx,y A(x,y)
S = cvAvg(A); // S = mean(A)
cvAvgStdDev(A,&m,&sigma); // compute mean
                          //  and standard deviation
a=cvNorm(A,B,norm); // a=||A-B||norm,
                    // norm is L1,L2,L∞
cvMinMaxLoc(A,&m,&M,&mpos,&Mpos); // find
   // global minimum, maximum and their positions
```

And there is a lot more!

See opencv/docs/ref/opencvref_cxcore.htm

Linear Algebra, Math Functions

cvGEMM(A,B,a,C,b,D,flags); // generalized matrix product: $D=a*A^{op(A)} *B^{op(B)} + b*C^{op(C)}$

cvSolve(A,b,x,method); // solve Ax=b (linear system or least squares problem)

cvSVD, cvSVDBkSb // singular value decomposition and back substitution

// some other functions to look at

cvDet, cvTrace, cvInvert, cvDotProduct, cvCrossProduct, cvEigenVV

cvDFT(A,B,flags); cvDCT(A,B,flags) // forward/inverse discrete Fourier and Cosine transforms

// fast vector transcendental functions (5-20x faster than exp(),log() etc.!)

cvExp(A,B); cvLog(A,B); cvPow(A,B,a); /* B=pow(A,a); */

cvCartToPolar(X,Y,mag,phase); cvPolarToCart(mag,phase,X,Y);

Dynamic Structures
(when 2d array is not enough)

The sample task: collect the positions of all non-zero pixels in the image.

Where to store the coordinates?

Possible solutions:

- Allocate array of maximum size (2*image_width*image_height – a huge value)
- Use two-pass algorithm
- Or use a list or similar data structure

```
// create a sequence of non-zero pixel positions
CvSeq* get_non_zeros( const IplImage* img, CvMemStorage* storage )
{
    CvSeq* seq = cvCreateSeq( CV_32SC2, sizeof(CvSeq), sizeof(CvPoint), storage );
    for( int i = 0; i < img->height; i++ ) for( int j = 0; j < img->width; j++ )
        if( CV_IMAGE_ELEM(img,uchar,i,j) )
     { CvPoint pt={j,i}; cvSeqPush( seq, &pt ); }
    return seq;
}
```

Memory Storages

Memory storage is a linked list of memory blocks.

It acts like stack:

new data is allocated always at the end

data is **not** deallocated until storage is deleted or cleared.

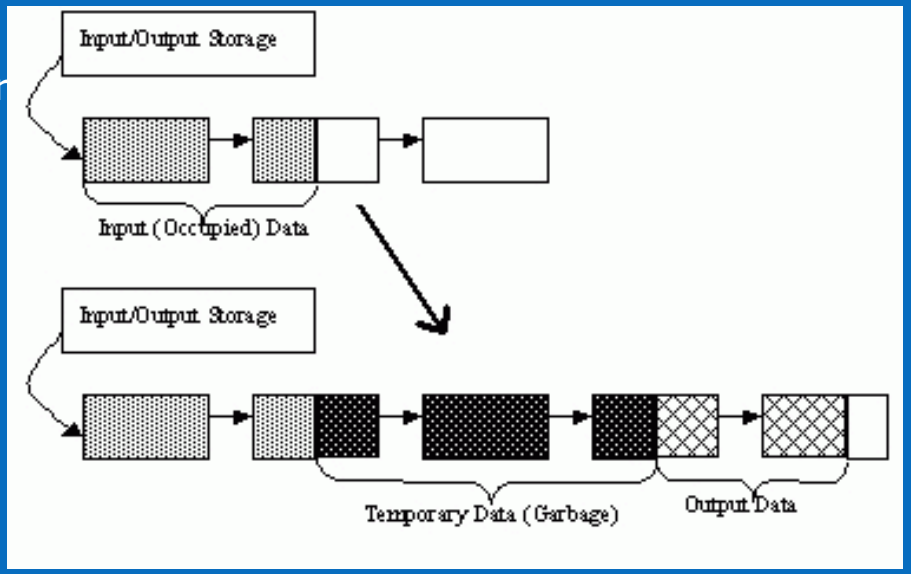
Key functions:

cvCreateMemStorage(block_size),

cvReleaseMemStorage(storage);

cvClearMemStorage(storage);

cvMemStorageAlloc(storage,size);



- In video/signal processing pipelines cvClearMemStorage is usually called in top-level loop.
- Good for multi-threaded applications
- Example: one of audio processing pipelines became several times faster when new/delete were replaced with CvMemStorage allocation functions.
- All OpenCV “dynamic structures” reside in memory storages

Sequences

CvSeq is a linked list of continuous blocks.

Large sequences may take multiple storage blocks, small sequences can be stored together in a single storage block (e.g. contours).

Sequence is deque: adding/removing elements to/from either end is O(1) operation,

inserting/removing elements from the middle is O(N).

Accessing arbitrary element is ~O(1)

Sequences can be sequentially read/written using readers/writers (similar to STL's iterators)

Sequences can not be released, only the whole storage can be deleted or cleared.

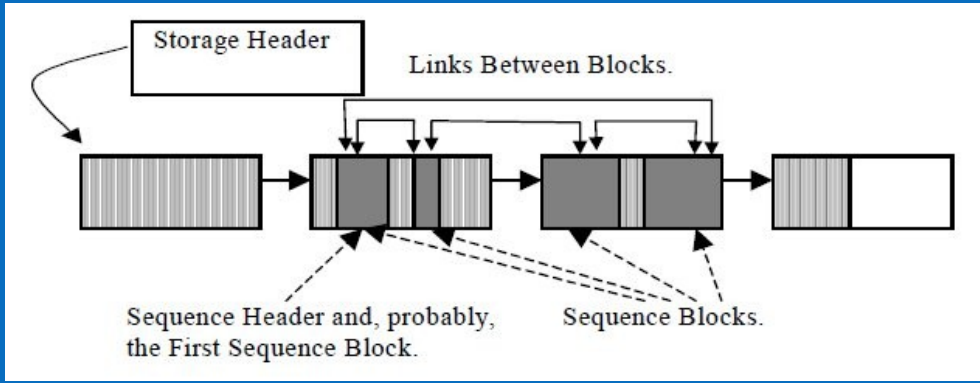
Key functions:

cvCreateSeq, cvClearSeq,

cvSeqPush, cvSeqPop, cvSetPushFront, cvSeqPopFront

cvGetSeqElem,

cvStartReadSeq, cvStartWriteSeq



Sets and Sparse Matrices

CvSet is unordered collection of elements, adding/removing elements is O(1) operation.

Sparse matrix in OpenCV uses CvSet + hash table to storing the elements.

```
// Collecting the image colors
CvSparseMat* get_color_map( const IplImage* img ) {
    int dims[] = { 256, 256, 256 };
    CvSparseMat* cmap = cvCreateSparseMat(3, dims, CV_32SC1);
    for( int i = 0; i < img->height; i++ ) for( int j = 0; j < img->width; j++ )
    {  uchar* ptr=&CV_IMAGE_ELEM(img,uchar,i,j*3);
       int idx[] = {ptr[0],ptr[1],ptr[2]};
       ((int*)cvPtrND(cmap,idx))[0]++; }
    // print the map
    CvSparseMatIterator it;
    for(CvSparseNode *node = cvInitSparseMatIterator( mat, &iterator );
        node != 0; node = cvGetNextSparseNode( &iterator )) {
        int* idx = CV_NODE_IDX(cmap,node); int count=*(int*)CV_NODE_VAL(cmap,idx);
        printf( "(b=%d,g=%d,r=%d): %d\n", idx[0], idx[1], idx[2], count ); }
    return cmap; }
```

Drawing Operations

It is possible to draw different shapes in an image (IplImage/CvMat):

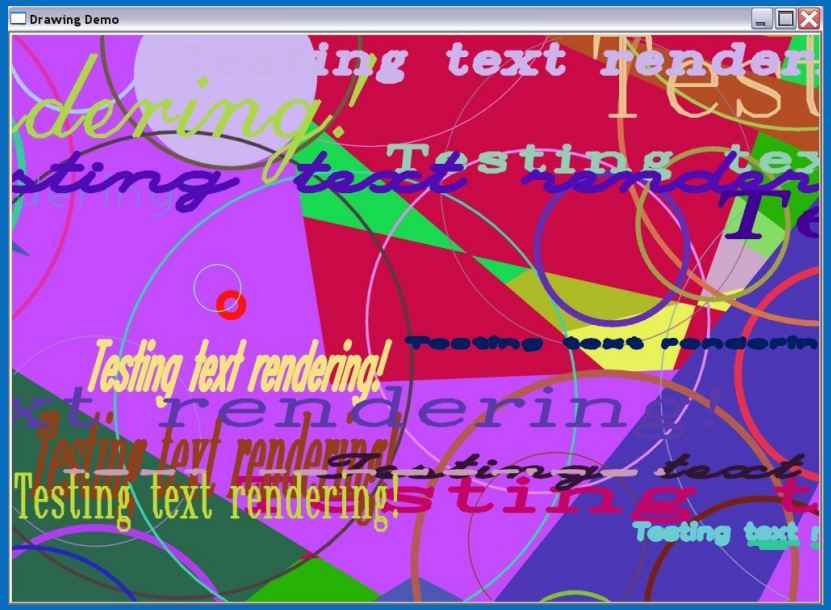
- Lines, Circles, Ellipses, Elliptic Arcs
- Filled polygons or polygonal contours
- Text (using one of embedded fonts)

Everything can be drawn with different colors, different line width, with or without antialiasing

Arbitrary images types are supported

All the rendering is done in memory, use cvShowimage to see the results immediately, use cvSaveImage to see the results later using external image viewer

opencv/samples/c/drawing.cpp:



Using CXCORE + HighGUI you can visualize everything,

from object tracking and vector fields to color histograms etc.

XML I/O: save your data

Need a config file?

Want to save results of your program to pass it to another one, or look at it another day?

Use cvSave, cvLoad and other XML I/O functions from cxcore.

```
// somewhere deep in your code… you get 5x5
// matrix that you'd want to save
{ CvMat A = cvMat( 5, 5, CV_32F, the_matrix_data );
  cvSave( "my_matrix.xml", &A );
}

// to load it then in some other program use …
CvMat* A1 = (CvMat*)cvLoad( "my_matrix.xml" );
```

my_matrix.xml

```
<?xml version="1.0"?>
<opencv_storage>
<my_matrix type_id="opencv-matrix">
  <rows>5</rows>
  <cols>5</cols>
  <dt>f</dt>
  <data>
    1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.
    0. 1.</data></my_matrix>
</opencv_storage>
```

So what about config file?

Writing config file

```
CvFileStorage* fs = cvOpenFileStorage("cfg.xml",
    0, CV_STORAGE_WRITE);
cvWriteInt( fs, "frame_count", 10 );
cvWriteStartWriteStruct( fs, "frame_size",
    CV_NODE_SEQ);
cvWriteInt( fs, 0, 320 );
cvWriteint( fs, 0, 200 );
cvEndWriteStruct(fs);
cvWrite( fs, "color_cvt_matrix", cmatrix );
cvReleaseFileStorage( &fs );
```

cfg.xml

```
<?xml version="1.0"?>
<opencv_storage>
<frame_count>10</frame_count>
<frame_size>320 200</frame_size>
<color_cvt_matrix type_id="opencv-
    matrix">
  <rows>3</rows> <cols>3</cols>
  <dt>f</dt>
  <data>…</data></color_cvt_matrix>
</opencv_storage>
```

Reading config file

```
CvFileStorage* fs = cvOpenFileStorage("cfg.xml", 0, CV_STORAGE_READ);
int frame_count = cvReadIntByName( fs, 0, "frame_count", 10 /* default value */ );
CvSeq* s = cvGetFileNodeByName(fs,0,"frame_size")->data.seq;
int frame_width = cvReadInt( (CvFileNode*)cvGetSeqElem(s,0) );
int frame_height = cvReadInt( (CvFileNode*)cvGetSeqElem(s,1) );
CvMat* color_cvt_matrix = (CvMat*)cvRead(fs,0,"color_cvt_matrix");
cvReleaseFileStorage( &fs );
```

Modules Descriptions: CV and CVAUX

Functionality Overview

Basic Image Processing: Image filters, Morphology, Pyramids, Color space conversions, Geometrical transformations, Histograms.

Advanced Image Processing and Feature extraction: Corner detection, Canny edge detector, Hough transform, Distance transform, Watershed, Inpainting, …

Shape analysis and Computational geometry: Image moments, contours, Delaunay triangulation and Voronoi Tesselation

Motion Analysis: Optical flow, Object tracking (Meanshift & CAMSHIFT), Motion templates, Kalman filters

Camera calibration, Epipolar geometry

Object detection (Haar classifier)

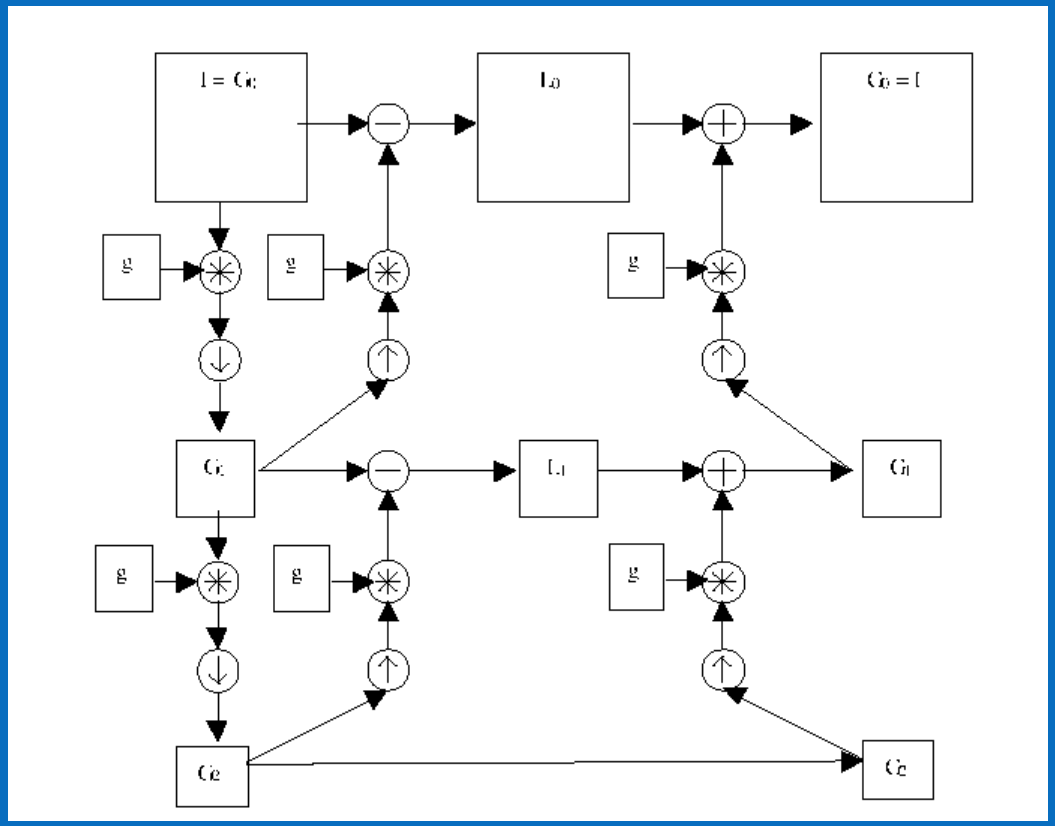
Advanced Blob tracking (aka Video Surveillance module)

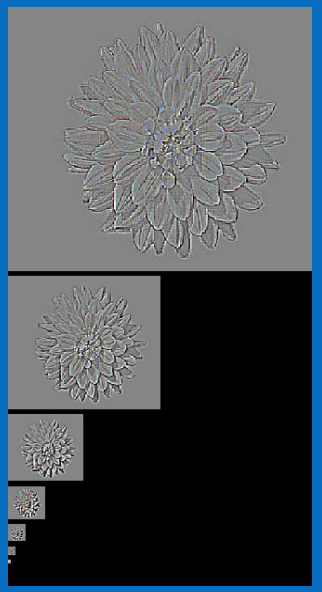
…

Basic Image Processing: Gaussian Pyramids

cvPyrDown(A,B); A – W×H, B – W/2×H/2

cvPyrUp(B,A1); B – W×H, A – 2W×2H





Basic Image Processing: Morphology

Primitive operations: cvErode(A,C,B); cvDilate(A,C,B);

Erosion: $C(x,y)=\min_{B(x',y')=1} A(x+x',y+y')$

Dilation: $C(x,y)=\max_{B(x',y')=1} A(x+x',y+y')$

Image A

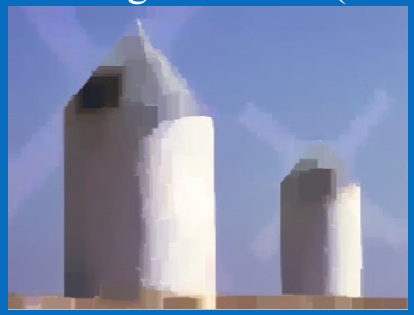
Erosion: C=AΘB

Dilation: C=A⊕B

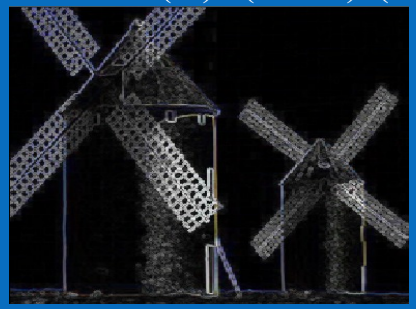
Compound operations: cvMorphologyEx(A,C,B,op);

Opening: C=AoB= (AΘB)⊕B

Closing: C=A•B= (A⊕B)ΘB



C=Grad(A)=(A⊕B)-(AΘB)



Basic Image Processing: Some other important functions

Color space conversion: cvCvtColor(A,B,color_conversion_code);

```
cvCvtColor(bgr_image, grayscale_image, CV_BGR2GRAY);
cvCvtColor(lab_image, bgr_image, CV_Lab2BGR);
```

Smoothing image using different methods: cvSmooth(A,B,method,parameters…);

```
cvSmooth(image, image, CV_GAUSSIAN, 7, 7, 1., 1.); // inplace Gaussian smoothing
                                                   // with 7x7 kernel and σ=1.
```

Computing spatial image derivatives: cvSobel(A,B,dx,dy,kernel_size);

```
cvSobel(A, Ax, 1, 0, 3); // compute dA/dx using 3x3 Sobel kernel:
```

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Image convolution with arbitrary kernel: cvFilter2D(A,B,kernel,anchor_point);

```
CvMat* kernel = cvCreateMat(11,11,CV_32F); cvSet(kernel,cvScalar(1./(kernel->rows*kernel->cols)));
cvFilter2D(A, B, kernel, cvPoint(kernel->cols/2,kernel->rows/2)); // slower alternative to
      cvSmooth(…, CV_BLUR,…)
```

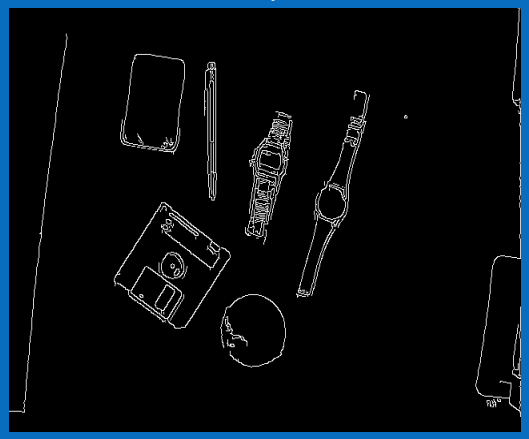
Computing image histogram: cvCalcArrHist(image_planes,hist);

```
int n = 256; CvHistogram* hist = cvCreateHist(1,&n,CV_HIST_ARRAY);
cvCalcArrHist((CvArr**)&grayscale_image, hist); // compute histogram for grayscale image
```

Feature Extraction: Canny Edge Detector

Computing binary edge map from grayscale image: cvCanny(A,B,t_1,t_2);

Assisting object segmentation

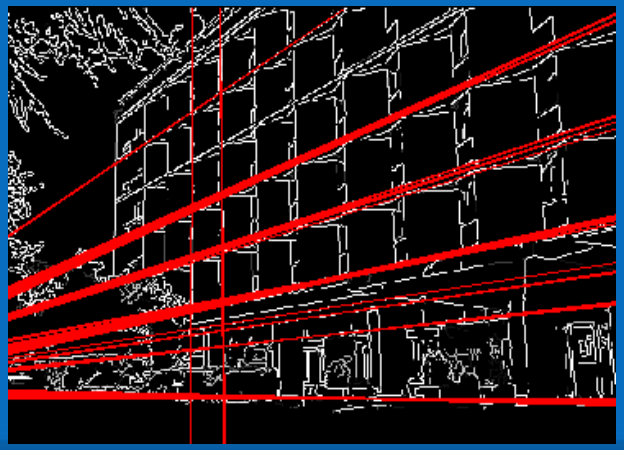


Accelerating face detection

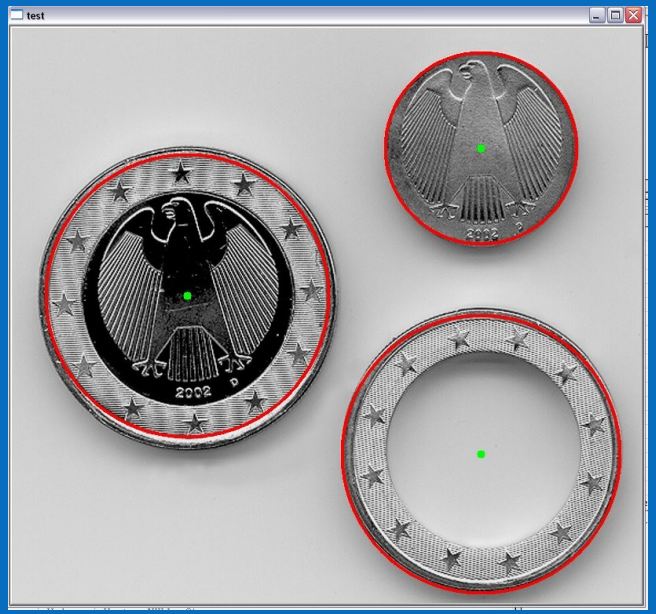
Feature Extraction: Hough Transform

Line Detection: cvHoughLines(image, lines_buffer, params)

Vanishing points estimation

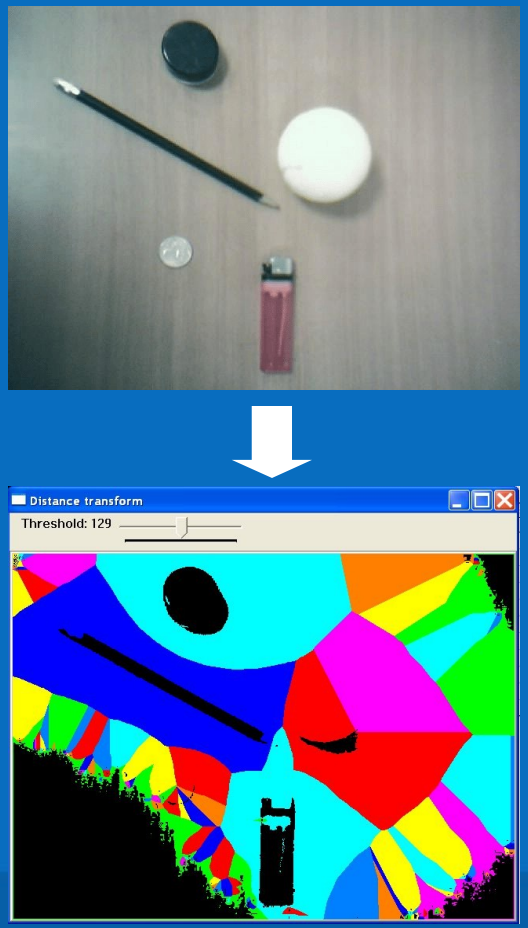


Circle Detection: cvHoughCircles(image, centers&radiuses)

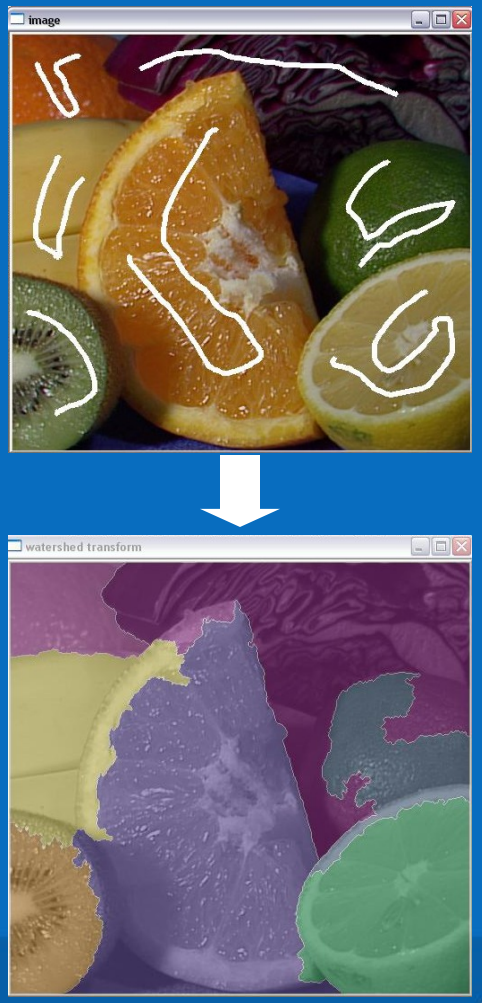


Advanced Image Processing: Extended Distance Transform and Watershed

Extended distance transform (O(N) Voronoi diagram):
cvDistTransform(A, distances,…, labels);



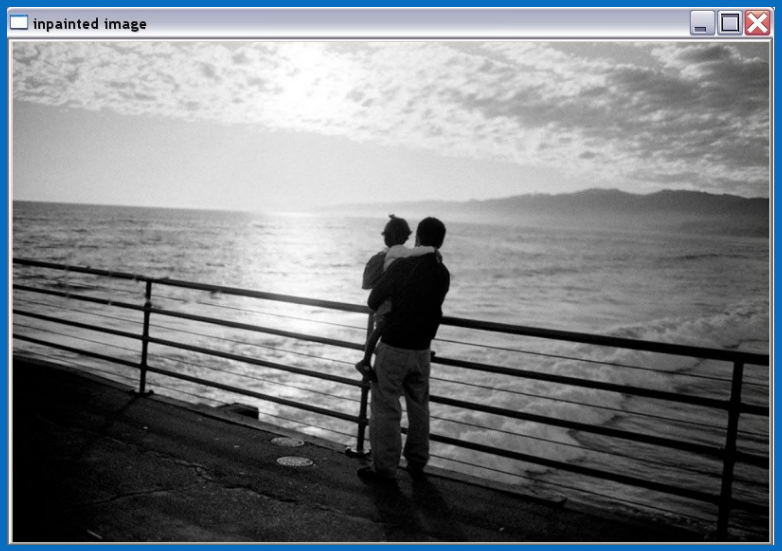
Watershed marker-based segmentation:
cvWatershed(A, markers);



Advanced Image Processing: Inpainting

Reconstruction of marked areas using the neighbor pixels:

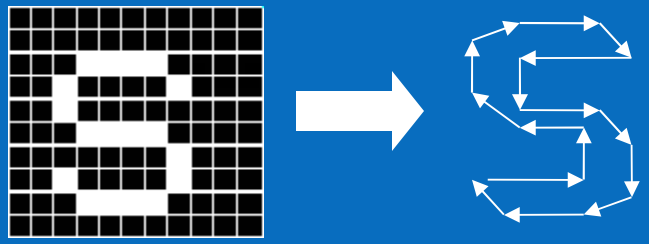
cvInpaint(image, mask);



Contours

In OpenCV contours are extracted from black-n-white images. Outer or internal boundaries of white areas are stored as polygons:

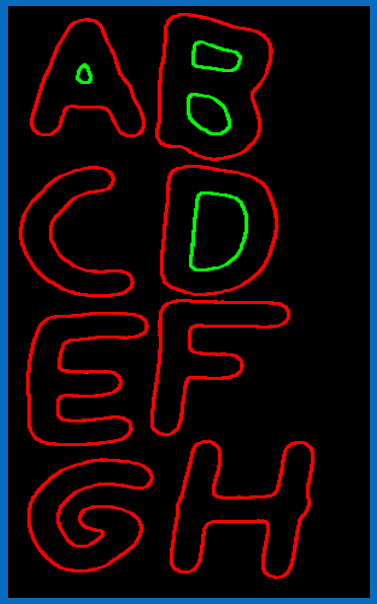
Contour representation:



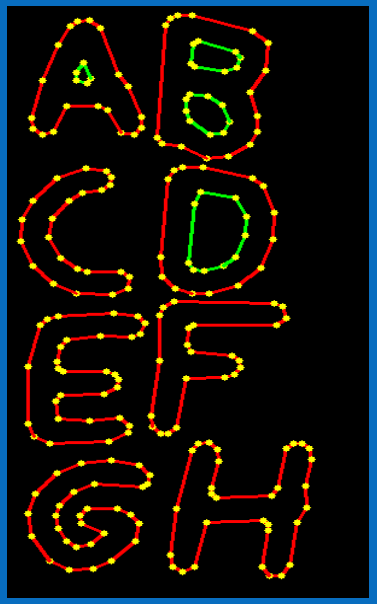
```
cvCmpS(grayscale, T, bw, CV_CMP_GT);
cvFindContours(bw, storage, &contours);
```

```
contours=cvApproxPoly(contours,params);
```

Grayscale image:
180000 pixels



Contours:
<1800 points



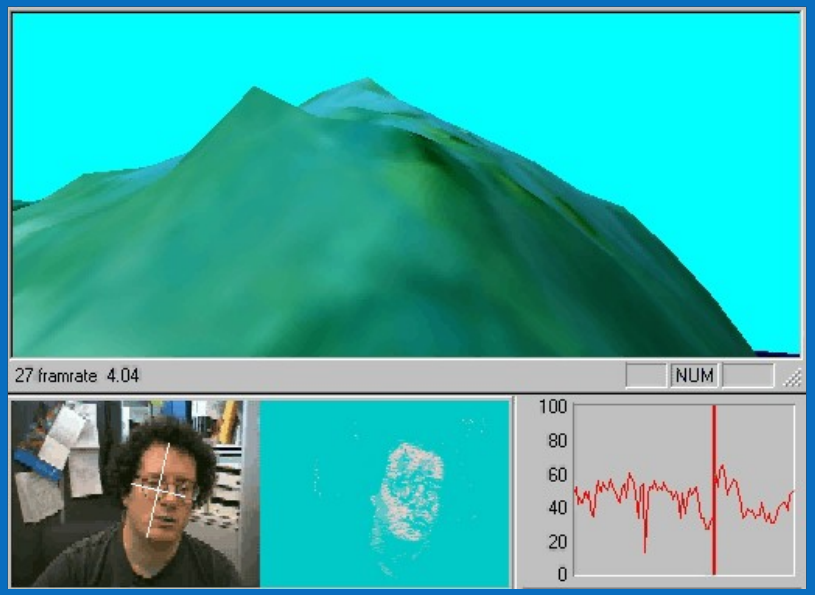
Approximated Contours:
<180 points

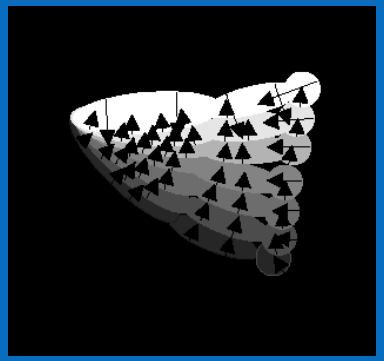
Motion Analysis and Object Tracking

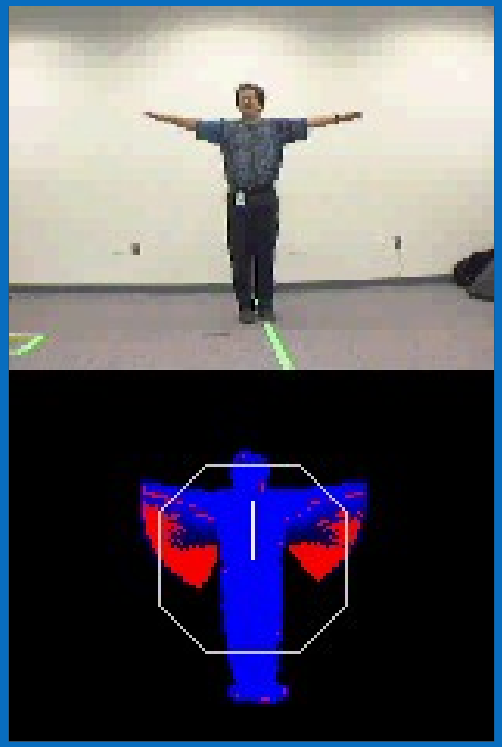
Feature tracking: cvCalcOpticalFlowPyrLK

Color object tracking: cvCamShift

Motion templates: cvUpdateMotionHistory, cvCalcMotionGradient, cvCalcGlobalOrientation, cvSegmentMotion

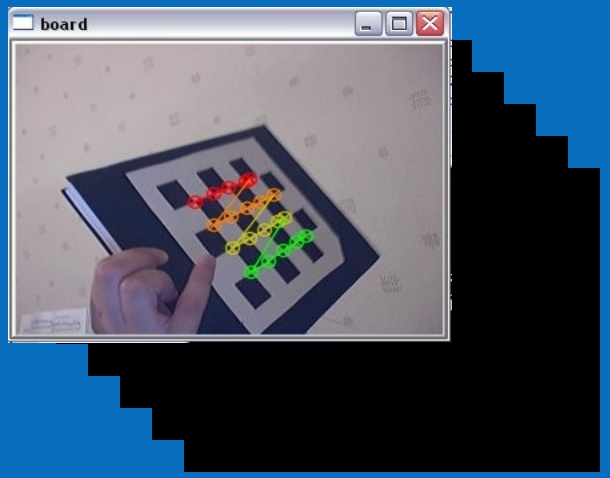






Camera Calibration

cvFindChessboardCorners

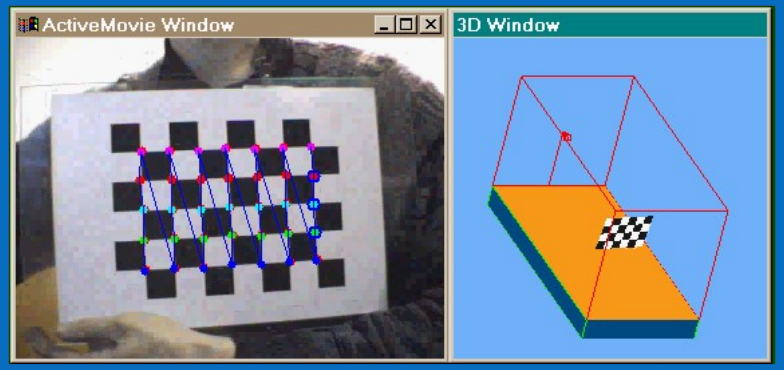


cvCalibrateCamera2

$$\left\{ \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \quad [k_1, k_2, p_1, p_2] \right\}$$

Camera matrix + lens distortion coefficients

cvUndistort2,
cvFindExtrinsicCameraParams2

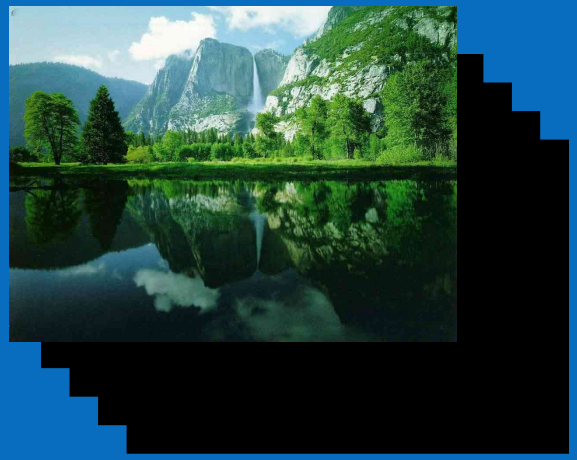


See opencv/samples/c/calibration.cpp

Face/Object Detection

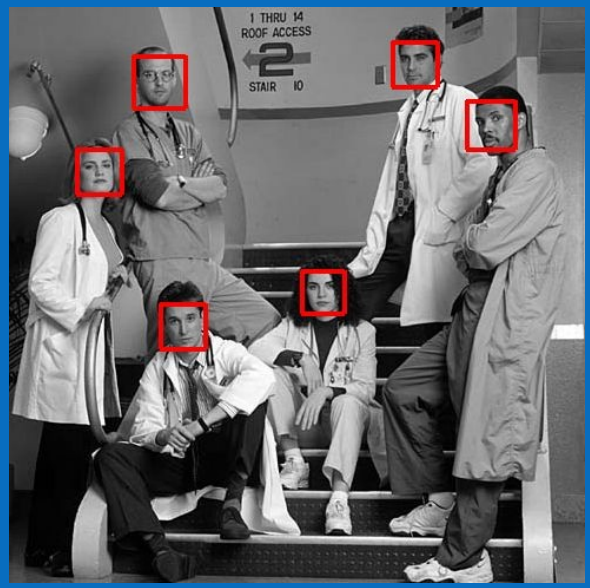
“objects”

“non-objects”



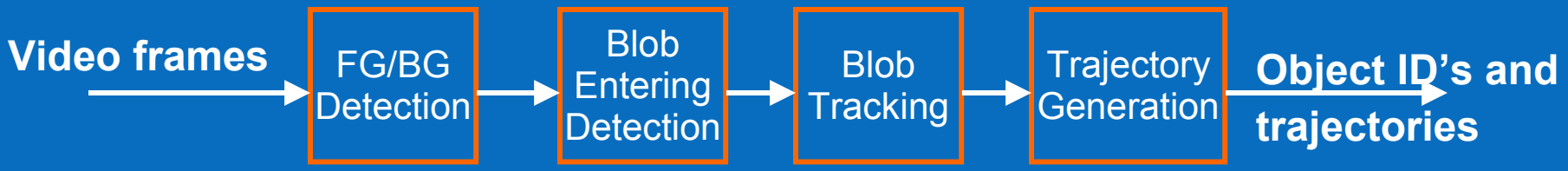
opencv/apps/haartraining

```
<?xml version="1.0" ?>
- <opencv_storage>
  - <haarcascade_frontalface_alt type_id="opencv-haar-classifier">
      <size>20 20</size>
    - <stages>
      - <_>
          <!-- stage 0 -->
        - <trees>
          - <_>
              <!-- tree 0 -->
            - <_>
                <!-- root node -->
              - <feature>
                - <rects>
                    <_>3 7 14 4 -1.</_>
                    <_>3 9 14 2 2.</_>
```



cvLoad, cvDetectHaarObjects

Video Surveillance



For details see opencv/docs/vidsurv, opencv/samples/c/blobtrack.cpp and opencv/cvaux/include/cvvidsurv.hpp

Modules Descrpitions: MLL

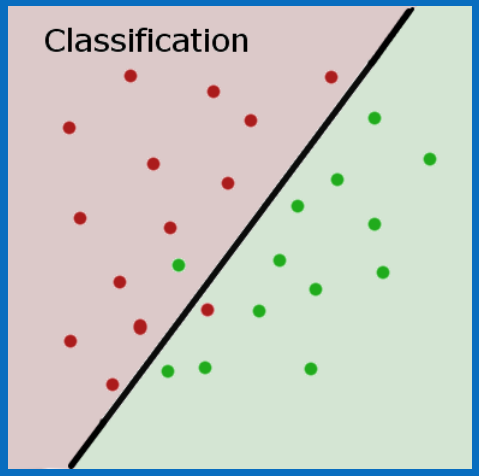
Functionality Overview

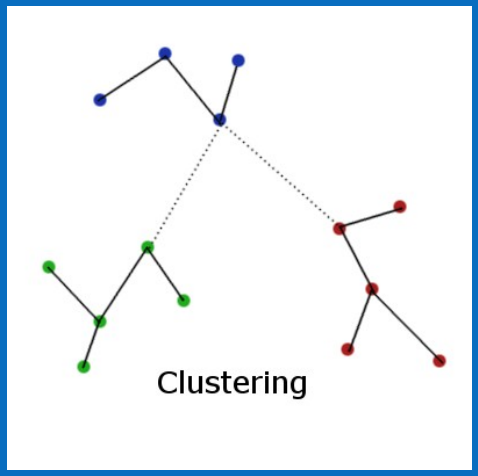
Statistical classification

Regression

Clustering

Algorithms: SVM, kNN, Normal Bayesian classifier, Neural Nets (MLP), Decision Trees, Random Trees, Boosting, EM.



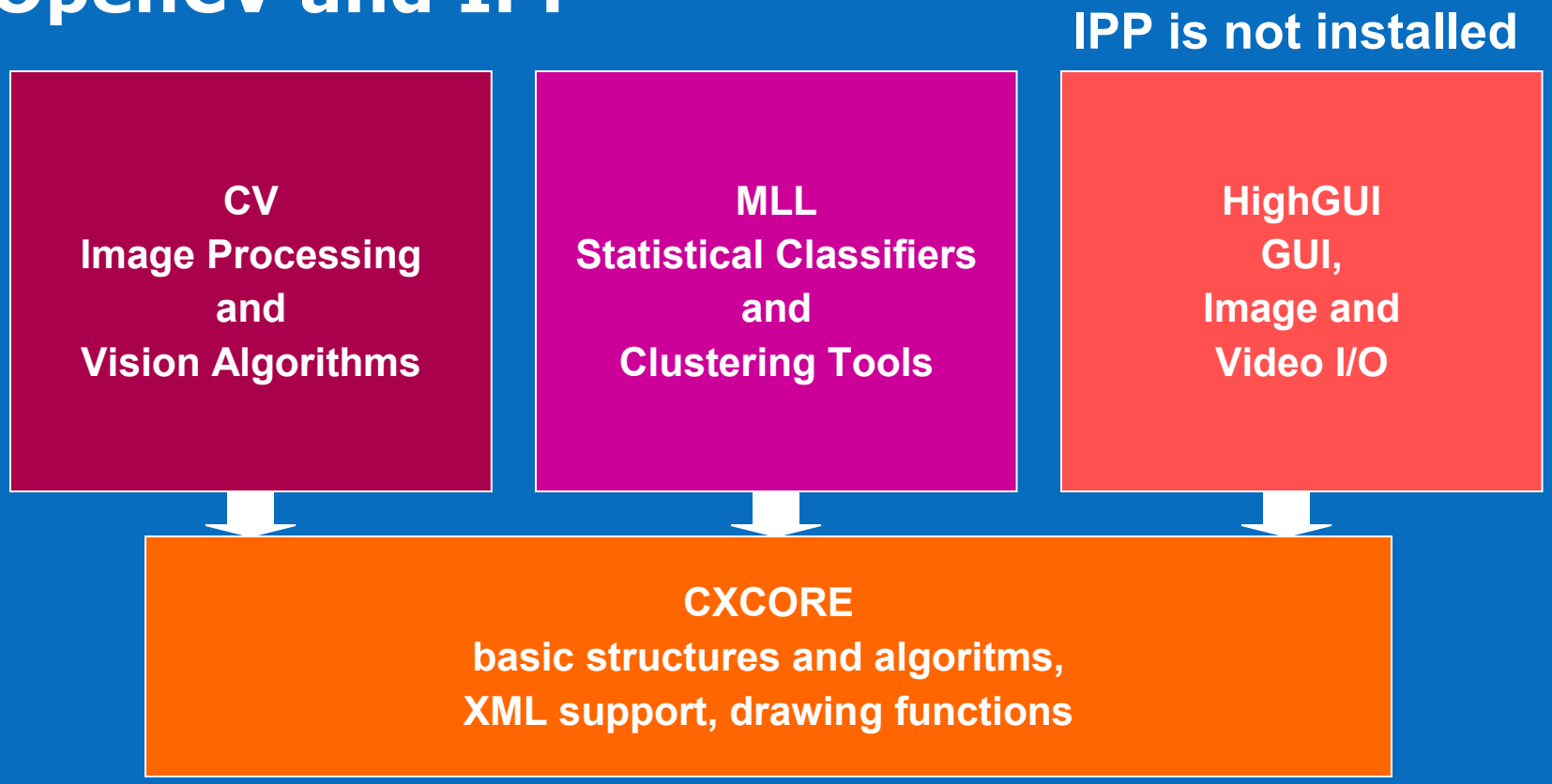


MLL Usage: SVM

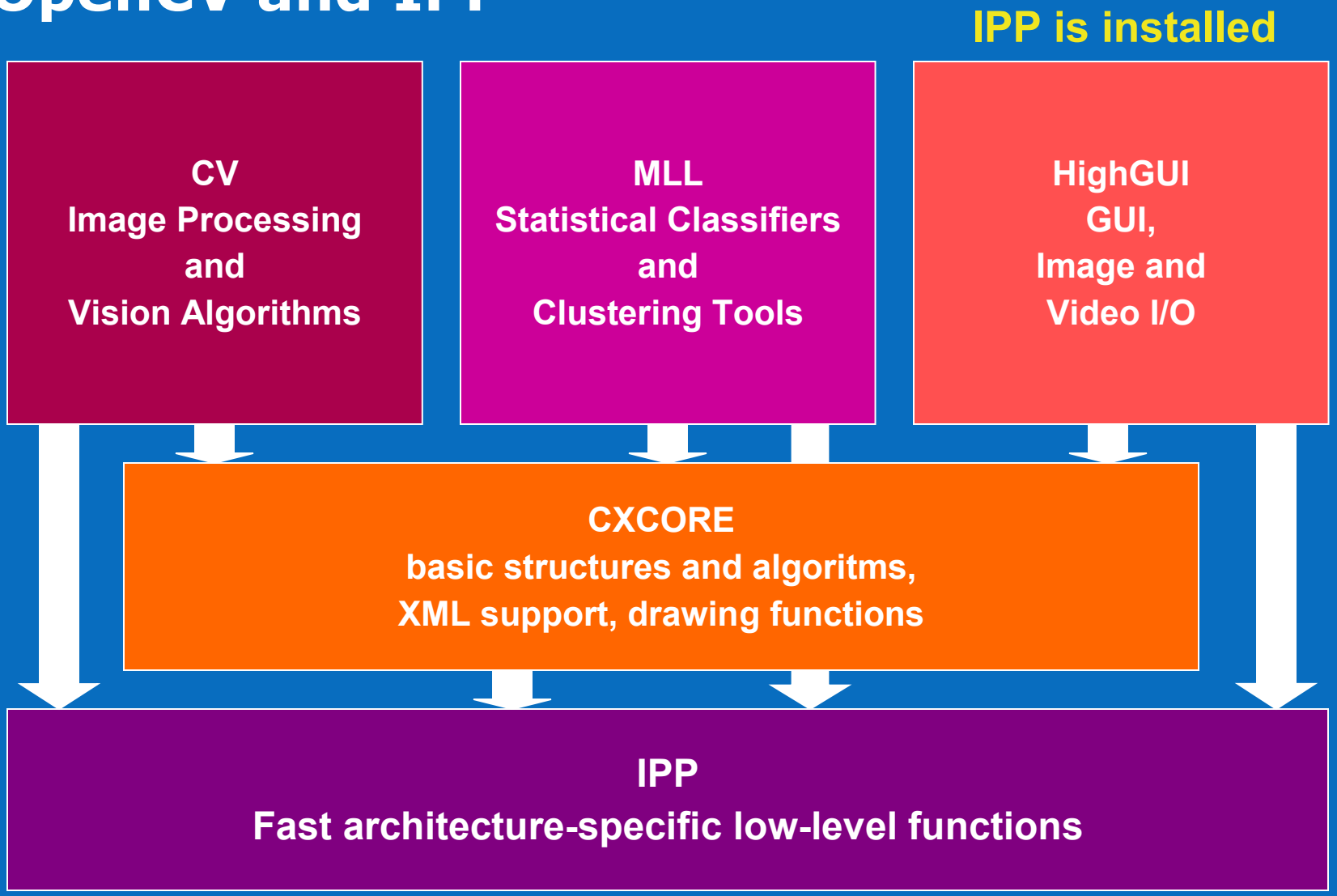
```
// … prepare matrix of training feature vectors, and store them as rows of matrix “data”,
// and prepare the vector of responses.
CvMat* data = cvCreateMat( num_samples, num_features, CV_32F );
CvMat* responses = cvCreateMat( num_samples, 1, CV_32F );
for(i=0;i<num_samples;i++) {
    for(j=0;j<num_features;j++) CV_MAT_ELEM(*data,float,i,j)=f_ij;
    CV_MAT_ELEM(*responses,float,i,0)=y_i;
}
// initialize parameters
CvSVMParams params(CvSVM::C_SVC, CvSVM::RBF, 0, myGamma, 0, myC, 0, 0,
   cvTermCriteria(CV_TERMCRIT_EPS+CV_TERMCRIT_ITERS, 1000, 1e-6));
// create and train the model
CvSVM svm(data, responses, 0, 0, params);

CvMat sample; cvGetRows(data,&sample,0,1);
float r = svm.predict(&sample); // predict the response for the first training vector
svm.save(“my_svm.xml”); // save the model to XML file
…
CvSVM svm1;
svm1.load(“my_svm.xml”); // load the model from XML file
```

Interaction with IPP

OpenCV and IPP



OpenCV and IPP



Performance Boost with IPP

Function Group	Speed increase (OpenCV+IPP vs pure OpenCV)
Gaussian Pyramids	~3x
Morphology	~3x-7x
Median filter	~2x-18x
Linear convolution	~2x-8x
Template Matching	~1.5x-4x
Color Conversion (RGB to/from Grayscale,HSV,Luv)	~1x-3x
Image moments	~1.5x-3x
Distance transform	~1.5x-2x
Image affine and perspective transformations	~1x-4x
Corner detection	~1.8x
DFT/FFT/DCT	~1.5x-3x
Vector math functions (exp, log, sin, cos …)	3x-10x
Face detection	~1.5x-2x

OpenCV + IPP recipe

Install OpenCV; make sure to add the **bin** subdirectory to the system path

Install IPP; make sure to add the **bin** subdirectory to the system path

Enjoy faster OpenCV apps!

How to check whether IPP is used or not?

```
const char* plugin_info = 0;
cvGetModuleInfo(0,0,&plugin_info);
ipp_is_used = plugin_info != 0 && strstr(plugin_info,"ipp")!=0;
```

How to turn off IPP? And turn on it again?

```
cvUseOptimized(0); … cvUseOptimized(1); // the function is slow!
```

Using OpenCV+IPP at Expert Level

Triple 3x3 median filter

```
#include <cv.h>
#include <highgui.h>
#include <ipp.h> // only for direct call to ippiMedianFilter_8u_C3R!
#include <stdio.h>
int main(int,char**){
  const int M=3;
  IppiSize msz={M,M}; IppiPoint ma={M/2,M/2};
  IplImage* img=cvLoadImage("lena.jpg",1);
  IplImage* med1=cvCreateImage(cvGetSize(img),8,3);
  IplImage* med2=cvCloneImage(med1);
  int64 t0 = cvGetTickCount(),t1,t2;
  IppiSize sz = {img->width-M+1,img->height-M+1};
  double isz=1./(img->width*img->height);
  cvSmooth(img,med1,CV_MEDIAN,M); // use IPP via OpenCV interface
  t0=cvGetTickCount()-t0;
  cvUseOptimized(0); // unload IPP
  t1 = cvGetTickCount();
  cvSmooth(img,med1,CV_MEDIAN,M); // use C code
  t1=cvGetTickCount()-t1;
  t2=cvGetTickCount();
  ippiMedianFilter_8u_C3R( // use IPP directly
    &CV_IMAGE_ELEM(img,uchar,M/2,M/2*3), img->widthStep,
    &CV_IMAGE_ELEM(med1,uchar,M/2,M/2*3), med1->widthStep, sz, msz, ma );
  t2=cvGetTickCount()-t2;
  printf("t0=%.2f, t1=%.2f, t2=%.2f\n", (double)t0*isz,(double)t1*isz,(double)t2*isz);
  return 0; }
```

How does it works?

(Quite Similar to IPP dispatcher mechanism)

```
// cv/src/_cvipp.h
…
IPCVAPI_EX(CvStatus, icvFilterMedian_8u_C3R, “ippiFilterMedian_8u_C3R”,
        CV_PLUGINS1(CV_PLUGIN_IPPI), (const void* src, int srcstep, void* dst, int dststep,
        CvSize roi, CvSize ksize, CvPoint anchor ))
…
```

```
// cv/src/cvswitcher.cpp
…
#undef IPCVAPI_EX
#define IPCVAPI_EX() …
static CvPluginFuncInfo cv_ipp_tab[] = {
#undef _CV_IPP_H_
/* with redefined IPCVAPI_EX every function declaration turns to the table entry */
#include "_cvipp.h“
#undef _CV_IPP_H_
   {0, 0, 0, 0, 0}
};
CvModule cv_module( &cv_info );
…
```

```
// cv/src/cvsmooth.cpp
icvFilterMedian_8u_C3R_t icvFilterMedian_8u_C3R_p = 0;
void cvSmooth() { if( icvFilterMedian_8u_C3R_p ) { /* use IPP */ } else { /* use C code */… }
```

OpenCV + Python

Python Interface

- Python is a great tool for fast algorithm/demo prototyping (no variable declaration, fully automatic memory management, no compile phase)
- Wrappers are generated automatically using SWIG
- Installs automatically with OpenCV
- Covers full cxcore, cv, highgui (no MLL yet)

```
from opencv.cv import *
from opencv.highgui import *
cvNamedWindow("win", CV_WINDOW_AUTOSIZE)
cap=cvCreateFileCapture(“./Road.avi”)
img2 = temp = None
while 1:
  img = cvQueryFrame(cap)
  if not img: break
  if not img2:
    img2 = cvCloneImage(img)
    temp = cvCloneImage(img)
  cvFlip(img, img2, 0)
  cvMorphologyEx(img2, img2, temp, None,
    CV_MOP_GRADIENT, 2 );
  cvShowImage("win", img2)
  if cvWaitKey(30) >= 0: break
```

Some Usage Examples

1. Improving IPP MPEG4 Encoder

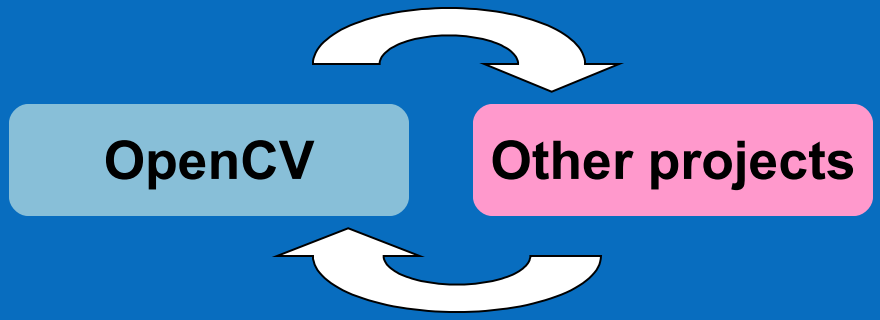
Problem statement: MPEG-4 has capability to store global motion transformation between frames => more compact representation of motion vectors

Want to estimate rigid affine transformation, robust to “outliers” – small objects moving in a different direction.

We just used cvCalcOpticalFlowPyrLK + RANSAC

With this feature IPP MPEG-4 video encoder (already very good one) achieved **10%** smaller bitrate with the same quality!

Now there is new function in OpenCV: cvEstimateRigidTransform



2. Extracting Text from Video

No ready function in OpenCV

What is text? How do we distinguish it from the other video content?

- Each letter has a single color (e.g. white or yellow)
- It has smooth shape
- Letters are organized in words: horizontally-aligned groups.
- All letters in the word (and all the subtitles in video frame) have the same color.
- Letters & words preserve their shapes across several frames, they either stay at the same position or move smoothly

- Let's use cvFindContours (we do not know the threshold level, so try different ones).
- Sort the contours by y coordinates of bounding rectangle to identify clusters (words)
- Use cvDrawContours + cvAvgStdDev to check the color.
- Track feature points of letter candidates using cvCalcOpticalFlowPyrLK and check motion smoothness and rigidity.

Summary

OpenCV is a long-term project and has established reputation because of good quality

Together with IPP it is used in different areas and for different kinds of applications, from small research projects to large commercial products.

Intel is still standing behind of OpenCV

OpenCV will continue to evolve; the further grow will be application-driven, (e.g. in video surveillance area)

Thank you!