

Getting Started with OpenCV

What is OpenCV?

- OpenCV stands for Open Source Computer Vision Library
- Being developed at Intel since 1999
- Written in C/C++; Contains over 500 functions.
- Available on Windows, Linux and MacOSX.
- Home page:
www.intel.com/technology/computing/opencv/
- Sourceforge page:
<http://sourceforge.net/projects/opencvlibrary>
- Yahoo user group:
<http://tech.groups.yahoo.com/group/OpenCV/>

Licensing

- **Extract from license:**

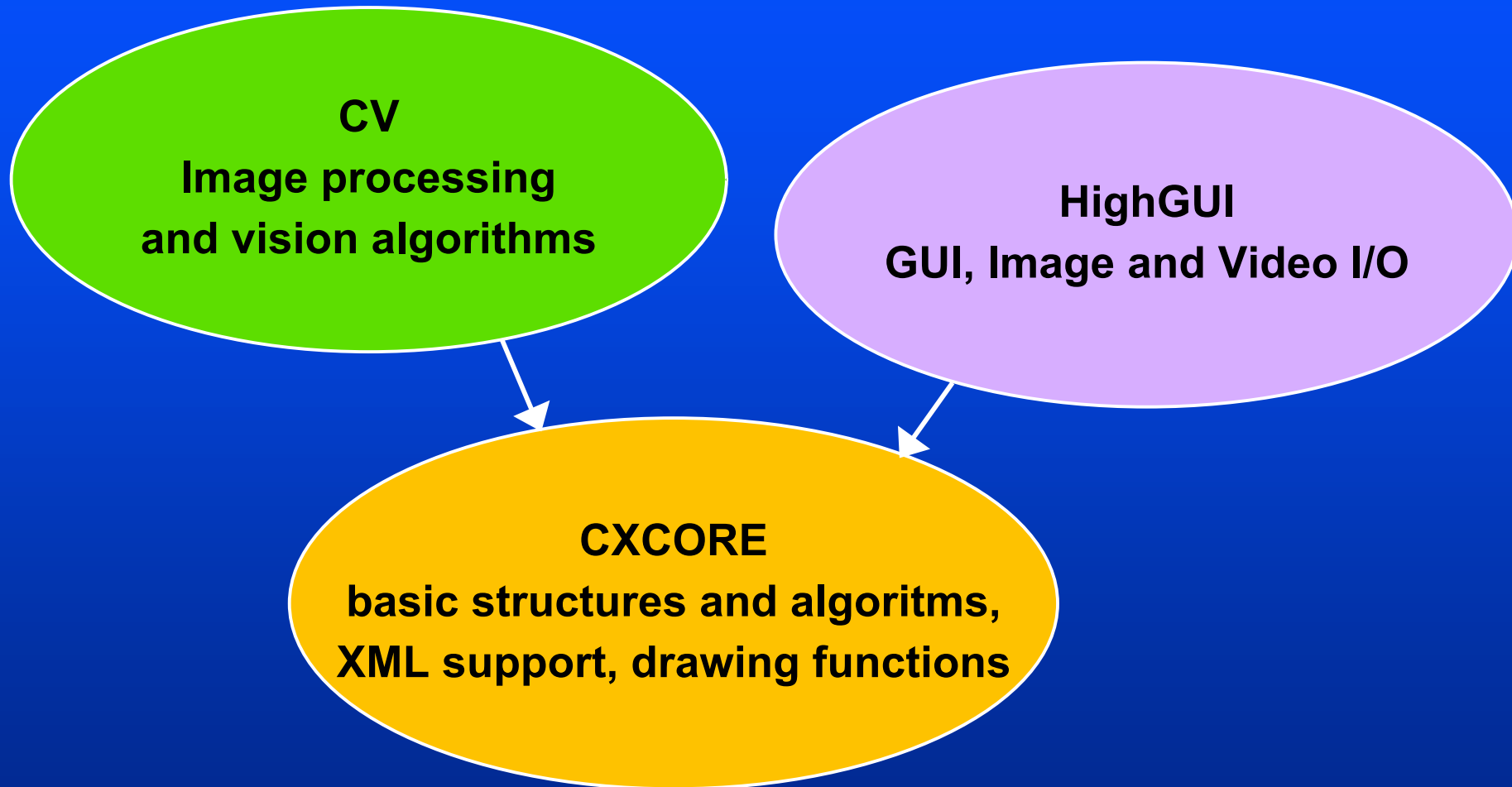
[Copyright Clause]

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistribution's of source code must retain the above copyright notice, this list of conditions and the following disclaimer.**
- * Redistribution's in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.**
- * The name of Intel Corporation may not be used to endorse or promote products derived from this software without specific prior written permission.**

[Disclaimer]

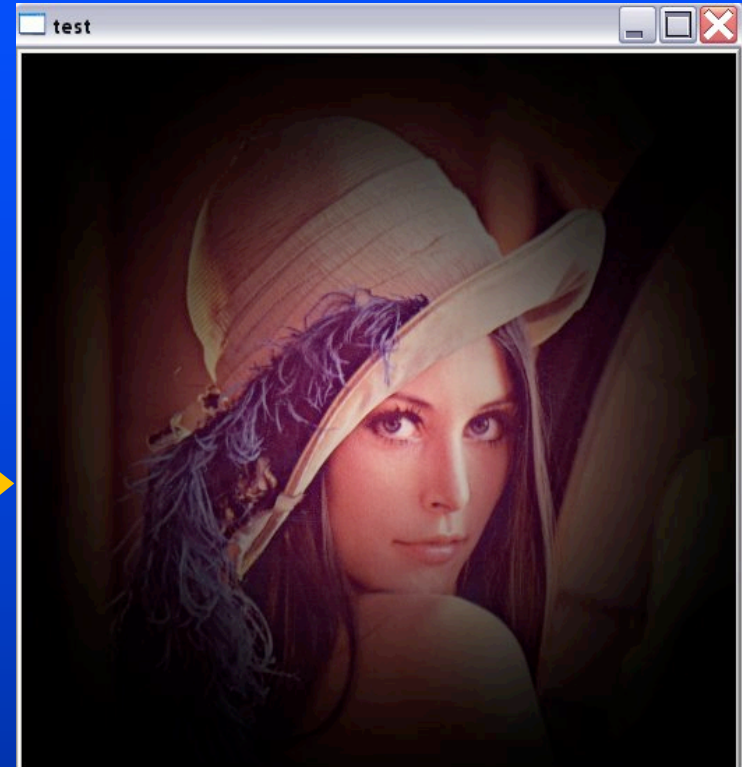
OpenCV structure



First OpenCV Program

```
1. #include <cxcore.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:

 Get the official release (currently, OpenCV b5 for Windows = OpenCV 0.9.7 for Linux) at <http://www.sourceforge.net/projects/opencvlibrary>

 Get the latest snapshot from CVS:

:pserver:anonymous@cvs.sourceforge.net:/cvsroot/opencvlibrary

Install it:

 On Windows run installer, on Linux use “./configure + make + make install” (see INSTALL document)

 On Windows: open opencv.dsw, do “build all”, add opencv/bin to the system path; on Linux - see a)

Build the sample:

● Windows:

- `cl /I<opencv_inc> test.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.dsp` as starting point

● Linux:

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

Run it:

`test lena.jpg` or
`./test lena.jpg`

The Program Quick Review

```
1. #include <cxcore.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; }
```

- **short and clear** program, no need to mess with MFC/GTK/QT/..., cross-platform
- **cvLoadImage()** and **cvSaveImage()** provide the easiest way to save/load images of various formats.
- **cvPoint** and other “constructor” functions make the code shorter and allow 1-line functions call.
- **CV_IMAGE_ELEM()** – pretty fast way to access image pixels
- **cvRound()** is very fast and convenient way to cast float/double to int
- **cvNamedWindow()** creates “smart” window for viewing an image
- **cvShowImage()** shows image in the window
- **cvWaitKey()** delays the execution until key pressed or until the specified timeout is over

Creating/deallocating images

```
IplImage* color = cvCreate(  
    cvSize(width,height),IPL_DEPTH_8U, 3);
```

To deallocate, you can then call
`cvReleaseImage(&image).`


```

void process(void* img) {

    IpImage* image = reinterpret_cast<IpImage*>(img);

    int nl= image->height;
    int nc= image->width * image->nChannels;
    int step= image->widthStep; // because of alignment

    // because imageData is a signed char*
    unsigned char *data=
        reinterpret_cast<unsigned char *>(image->imageData);

    for (int i=0; i<nl; i++) {
        for (int j=0; j<nc; j+= image->nChannels) {

            // 3 channels per pixel

            if (data[j+1] > data[j] && data[j+1] > data[j+2]) {

                data[j]= 0xFF; // 255
                data[j+1]= 0xFF;
                data[j+2]= 0xFF;

            }

        }

        data+= step; // next line
    }
}

```

Accessing image pixels

APIs Reference

- Cxcore, Cv, HighGui and CvAux API reference

Cxcore

Cv

HighGui

Cvaux

What else HighGUI can do?

- “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
Linux – GTK+ 2
MacOSX – X11 & GTK+ 2; Native Aqua support is planned.

Image I/O

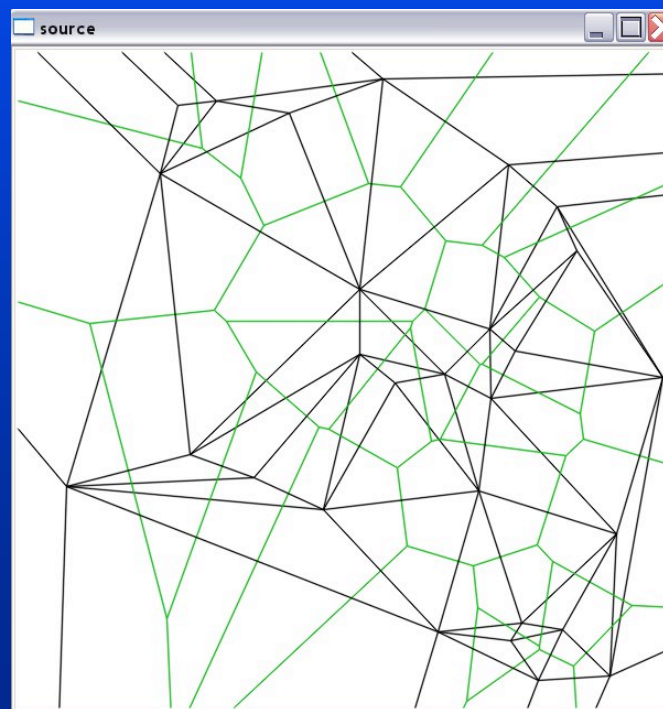
- **`IplImage* cvLoadImage(image_path, colorness_flag);`**
loads image from file, converts to color or grayscale, if need, and returns it (or returns NULL).
image format is determined by the file contents.
- **`cvSaveImage(image_path, image);`**
saves image to file, image format is determined from extension.
- BMP, JPEG (via libjpeg), PNG (via libpng), TIFF (via libtiff), PPM/PGM formats are supported.

```
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=0.
- The **only** function in highgui that process message queue => should be called periodically.
- Thanks to the function many highgui programs have clear sequential structure, rather than event-oriented structure, which is typical for others toolkits
- 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;
}
```



delaunay.c, 240 lines

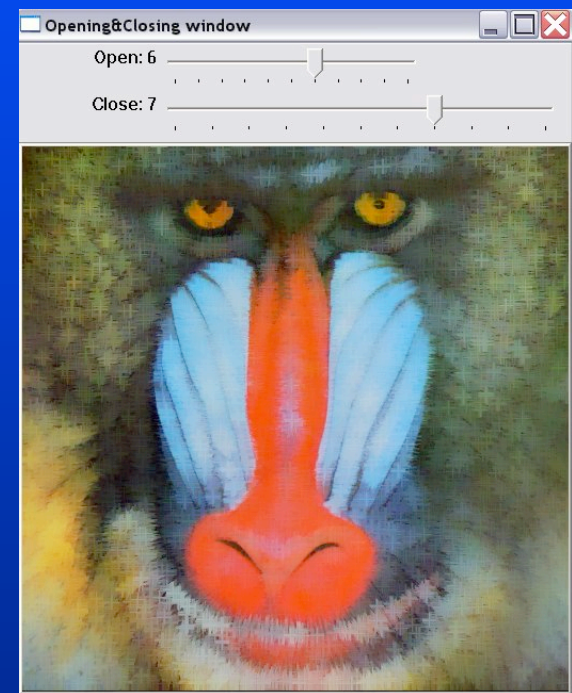
Trackbars

- `cvCreateTrackbar(trackbar_name,window_name, position_ptr,max_value,callback=0);`

creates trackbar and attaches it to the window.

Value range 0..max_value. When the position is changed, the global variable updated and callback is called, if specified.

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



morphology.c, 126 lines

Mouse Events

- **cvSetMouseCallback(window_name, callback, userdata=0);**
sets callback on mouse events (button clicks, cursor moves) for the specified window

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

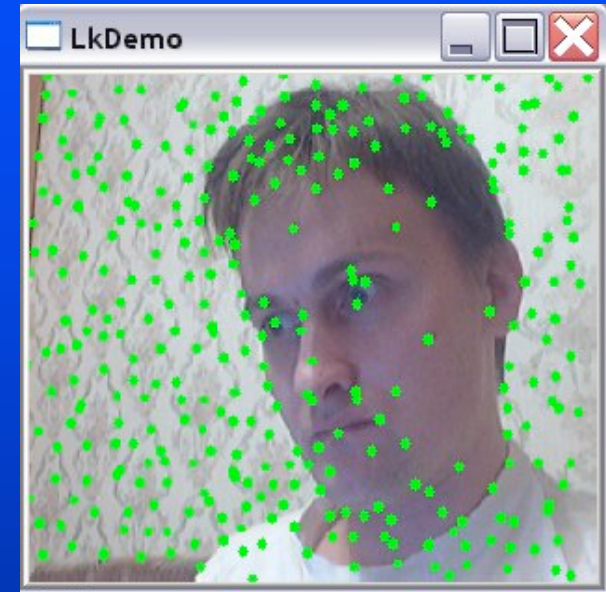
Scroll forward
for example 😊

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 occurred.
- **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 resources at the end!
- **Used interfaces:**
 - Windows: VFW, IEEE1394, MIL, DShow, Quicktime (in progress)
 - Linux: V4L2, IEEE1394, FFMPEG (nb: only 0.4.9-pre1 is working!)
 - MacOSX: FFMPEG, Quicktime (in progress)

Video I/O Example

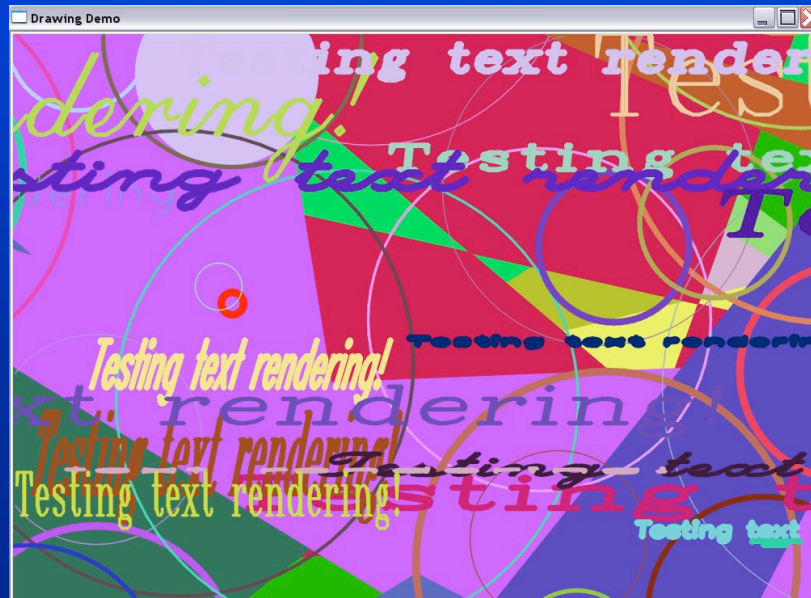
```
// opencv/samples/c/lkdemo.c
int main(...){
...
CvCapture* capture = <...> ?
    cvCaptureFromCAM(camera_id) :
    cvCaptureFromFile(path);
if( !capture ) return -1;
for(;;) {
    IplImage* frame=cvQueryFrame(capture);
    if(!frame) break;
    // ... copy and process image
    cvShowImage( "LkDemo",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



lkdemo.c, 190 lines
(needs camera to run)

What can be drawn in image (besides circles)?

- **cxcore provides numerous drawing functions:**
 - 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, antialiasing on/off
 - Arbitrary images types are supported (for depth!=8u antializing is off)



Dynamic Structures

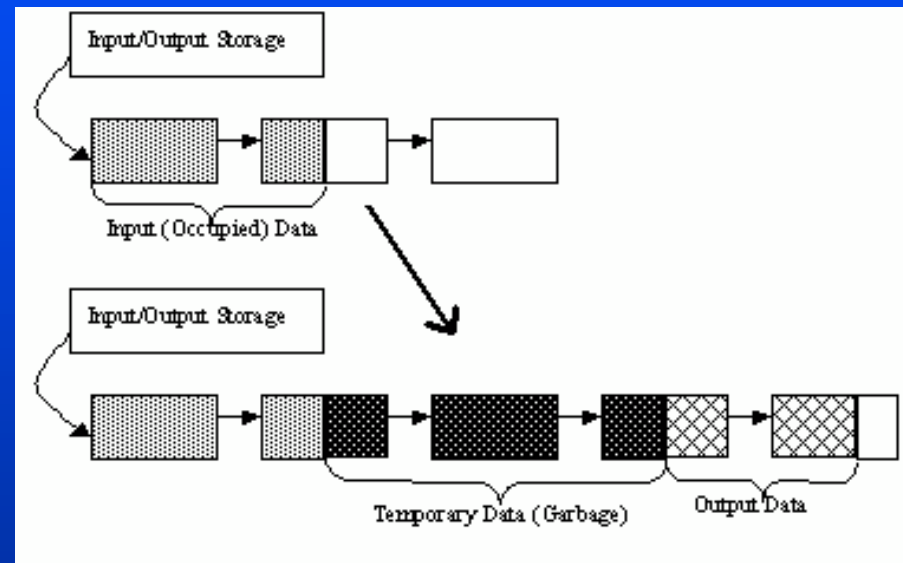
(when 2d array is not enough)

- The sample task: collect the locations of all non-zero pixels in the image.
- Where to store the locations? Possible solutions:
 - Allocate array of maximum size (`sizeof(CvPoint)*width*height` – a huge value)
 - Use two-pass algorithm
 - Use list or similar data structure
 - ... (Any other ideas?)

```
// construct sequence of non-zero pixel locations
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;
}
```

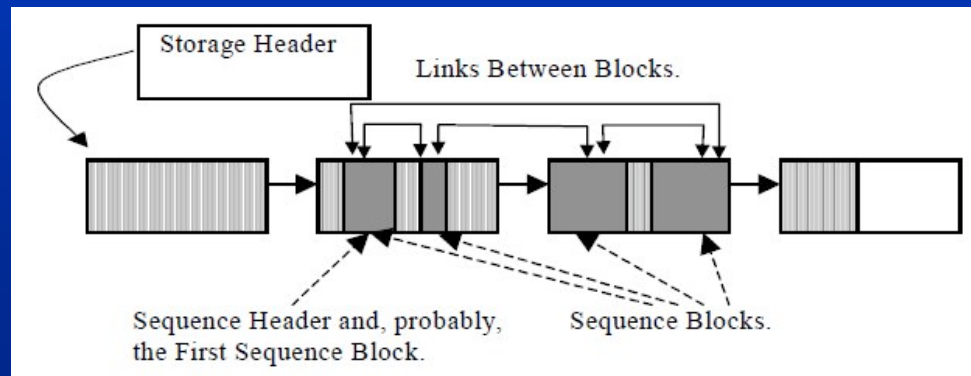
Memory Storages ...

- Memory storage is a linked list of memory blocks.
- Data is allocated in the storage by moving “free space” position, new blocks are allocated and added to the list if necessary
- It's only possible to free the continuous block of data by moving position back, thus both allocation and deallocation are virtually instant(!)
- Functions to remember:
`cvCreateMemStorage(block_size),`
`cvReleaseMemStorage(storage);`
`cvClearMemStorage(storage);`
`cvMemStorageAlloc(storage,size);`
- All OpenCV “dynamic structures” reside in memory storages.
- The model is very efficient for repetitive processing, where `cvClearMemStorage` is called on top level between iterations.



Sequences

- Sequence is a linked list of memory blocks (sounds familiar, eh?).
- Large sequences may be split into multiple storage blocks, while several small sequences may fit into the single storage block.
- 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 also $\sim O(1)$ (when storage block size \gg sequence block)
- Sequences can be sequentially read/written using readers/writers (similar to STL's iterators)
- Sequences can not be released, use `cv{Clear|Release}MemStorage()`
- Basic Functions to remember:
 - `cvCreateSeq(flags,header_size,element_size,storage),`
 - `cvSeqPush[Front](seq,element), cvSeqPop[Front](seq,element),`
 - `cvClearSeq(seq), cvGetSeqElem(seq,index),`
 - `cvStartReadSeq(seq,reader), cvStartAppendToSeq(seq,writer).`



Sets and Sparse Matrices

- Set is variant of sequence with “free node list” and it’s active elements do not follow each other – that is, it is convenient way to organize a “heap” of same-size memory blocks.
- Sparse matrix in OpenCV uses CvSet for storing 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; }
```

Save your data (to load it then)

- Need a config file?
- Want to save results of your program to pass it to another one, or look at it another day?

It all can be easily done with OpenCV
persistence-related functions.

my_matrix.xml

```
// 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" );
```

```
<?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 );
```

Some OpenCV Tips and Tricks

- Use short inline “constructor” functions: **cvScalar**, **cvPoint**, **cvSize**, **cvMat** etc.
- Use **cvRound** and vector math functions (**cvExp**, **cvPow** ...) for faster processing
- Use **cvStackAlloc** for small buffers
- Consider **CV_IMPLEMENT_QSORT_EX()** as possible alternative to **qsort()** & **STL sort()**.
- To debug OpenCV program when runtime error dialog error pops up, press “Retry” (works with debug version of libraries)

...

Alternatives

- VXL (C++, template based, Oxford+others, huge!)
- Gandalf (C, mainly low-level, LGPL)
- Matlab
- Lush (Lisp based, binding for opencv, opengl, SDL)
- Lots of commercial libraries
- Lots of small libraries dedicated to specific tasks