TrabHCI

Technologies to Reduce the Access Barrier in Human Computer Interaction

Erasmus Intensive Programme IP29588-1-1731-10

Image processing for gesture recognition: from theory to practice

1

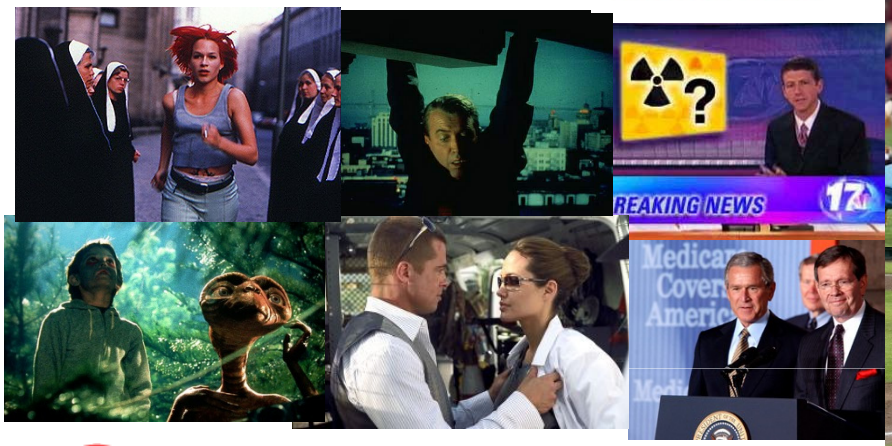
Michela Goffredo
University Roma TRE
goffredo@uniroma3.it

What's Computer Vision?

Why study Computer Vision?

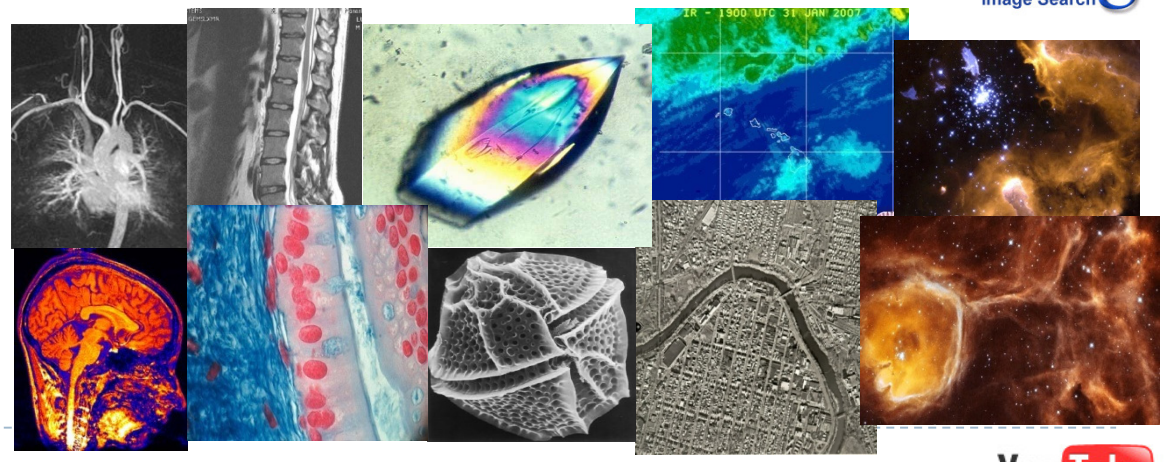
- Images and video are everywhere!

Personal photo albums



Movies, news, sports

Surveillance and security

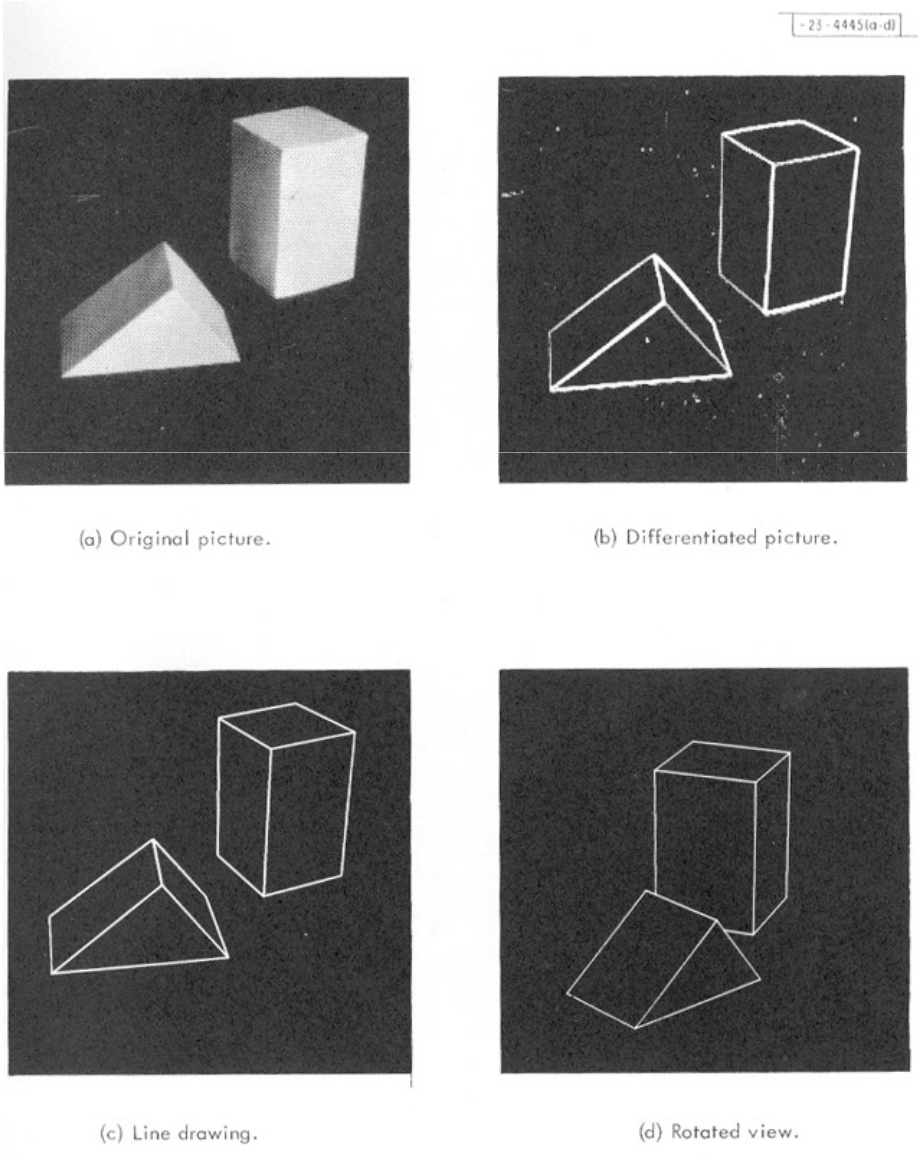


Medical and scientific images

What's Computer Vision?

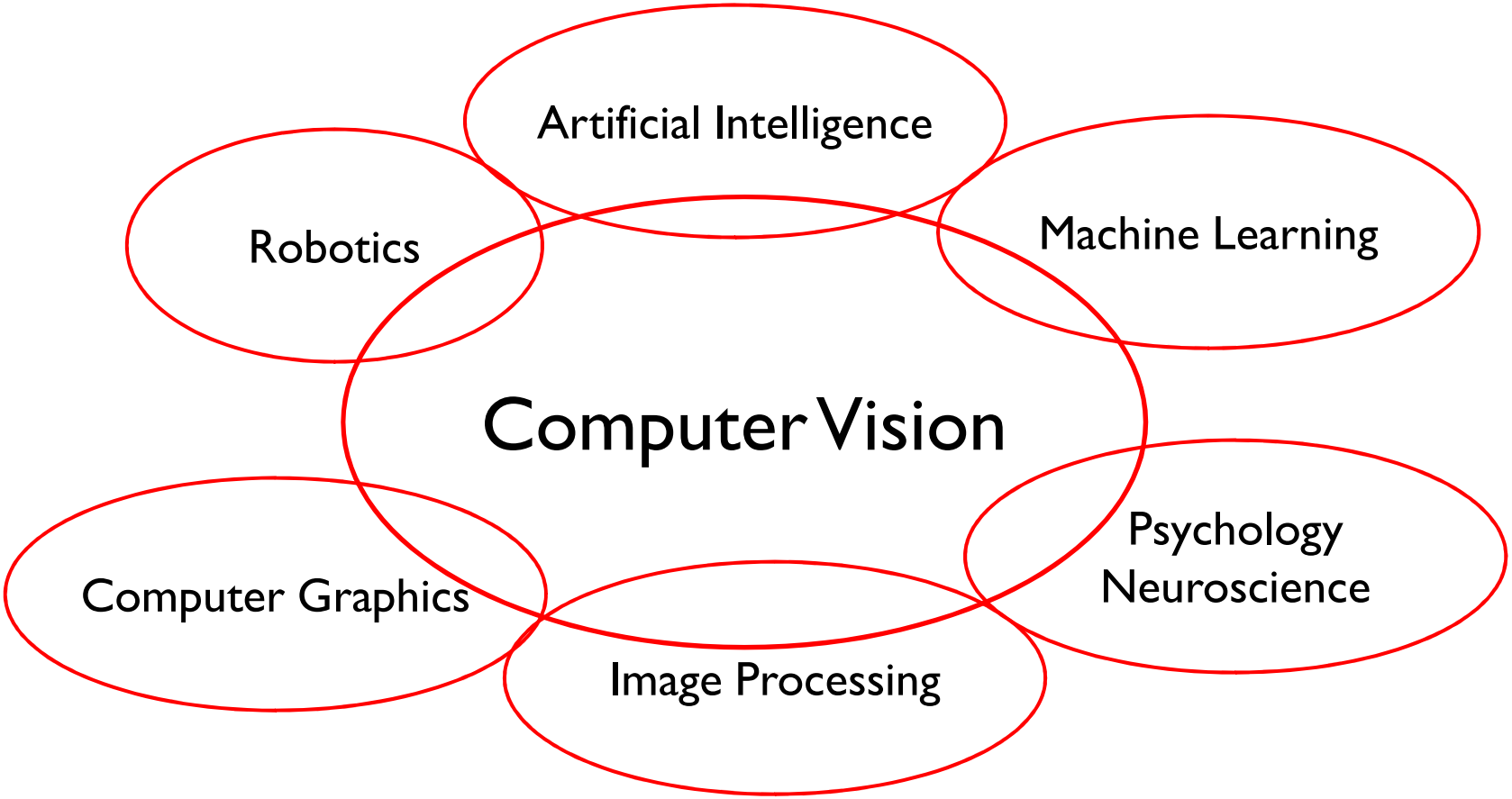
- Computer vision is the **transformation of data** from a still or video camera into either a decision or a new representation.
- All such transformations are done for achieving some particular goal.
- The input data may include some **contextual information** such as "the camera is mounted in a car" or "laser range finder indicates an object is one meter away".
- The decision might be "there is a person in this scene" or "there are 14 tumor cells on this slide".

Origins of computer vision



L. G. Roberts, *Machine Perception of Three Dimensional Solids*, Ph.D. thesis, MIT Department of Electrical Engineering, 1963.

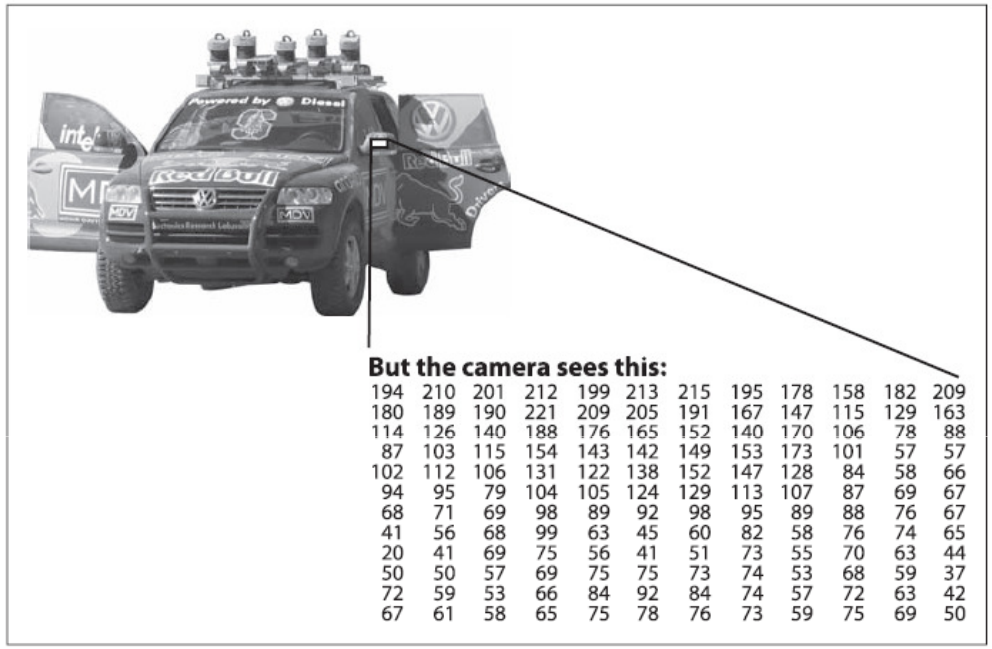
Connections to other disciplines



What's Computer Vision?

- How hard can it be to find, say, a car when you are staring at it in an image?
- The human brain divides the vision signal into many channels that stream different kinds of information into your brain. Your brain has an attention system that identifies, in a task-dependent way, important parts of an image to examine while suppressing examination of other areas. There is massive feedback in the visual stream that is, as yet, little understood.
- There are widespread associative inputs from muscle control sensors and all of the other senses that allow the brain to draw on cross-associations made from years of living in the world. The feedback loops in the brain go back to all stages of processing including the hardware sensors themselves (the eyes), which mechanically control lighting via the iris and tune the reception on the surface of the retina.

Perceive the “world behind the picture”



- In a machine vision system, however, a computer receives a grid of numbers from the camera or from disk, and that’s it.
- Data is corrupted by noise and distortions: from variations in the world (weather, lighting, reflections, movements), imperfections in the lens and mechanical setup, finite integration time on the sensor (motion blur), electrical noise in the sensor or other electronics, and compression artifacts…

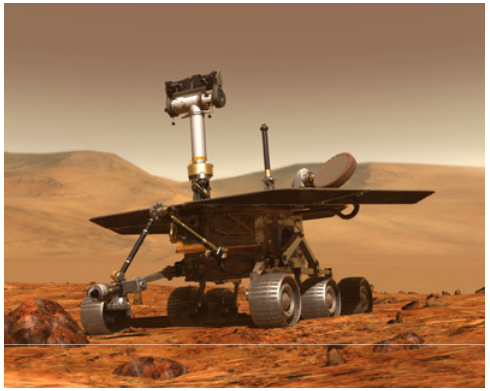
What's Computer Vision?

- In the design of a practical system, additional contextual knowledge can often be used to work around the limitations imposed on us by visual sensors.
- The actions or decisions that computer vision attempts to make based on camera data are performed in the context of a specific purpose or task.
- General rule: the more constrained a computer vision context is, the more we can rely on those constraints to simplify the problem and the more reliable our final solution will be.

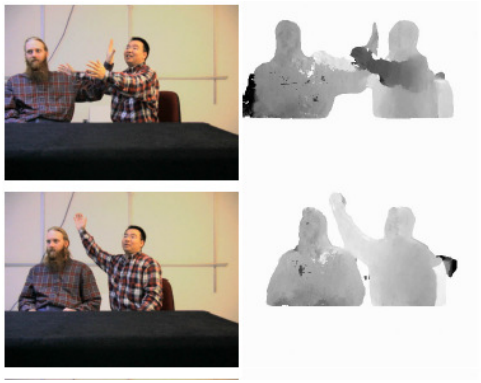
- What exactly does this mean?
  - Vision as a source of metric 3D information
  - Vision as a source of semantic information

Vision as measurement device

Real-time stereo

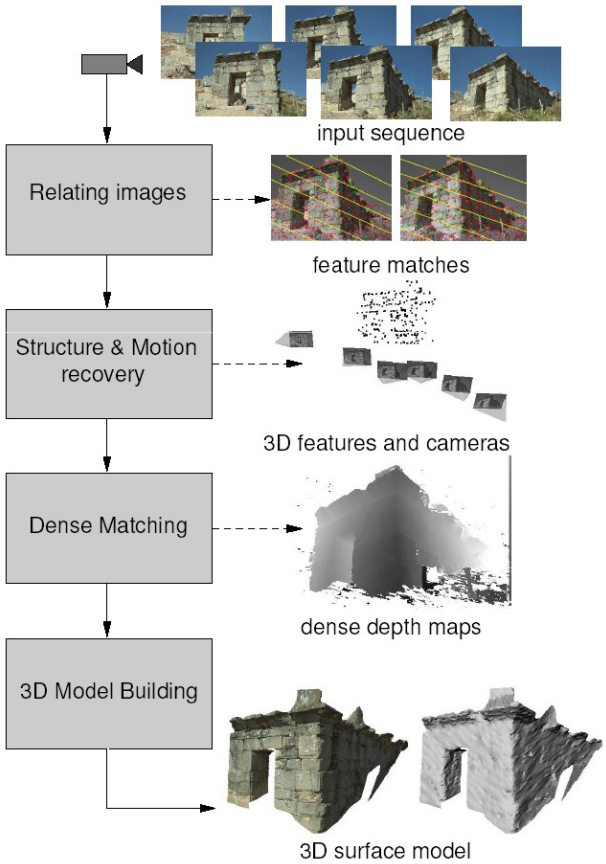


NASA Mars Rover

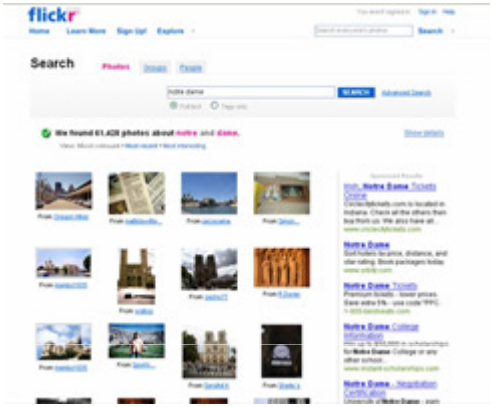


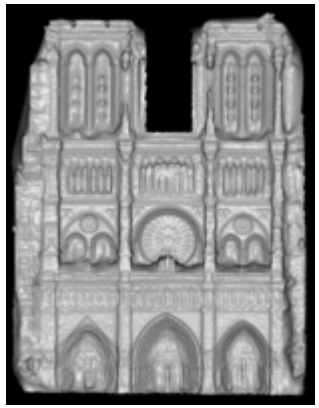
Pollefeys et al.

Structure from motion



Multi-view stereo for community photo collections

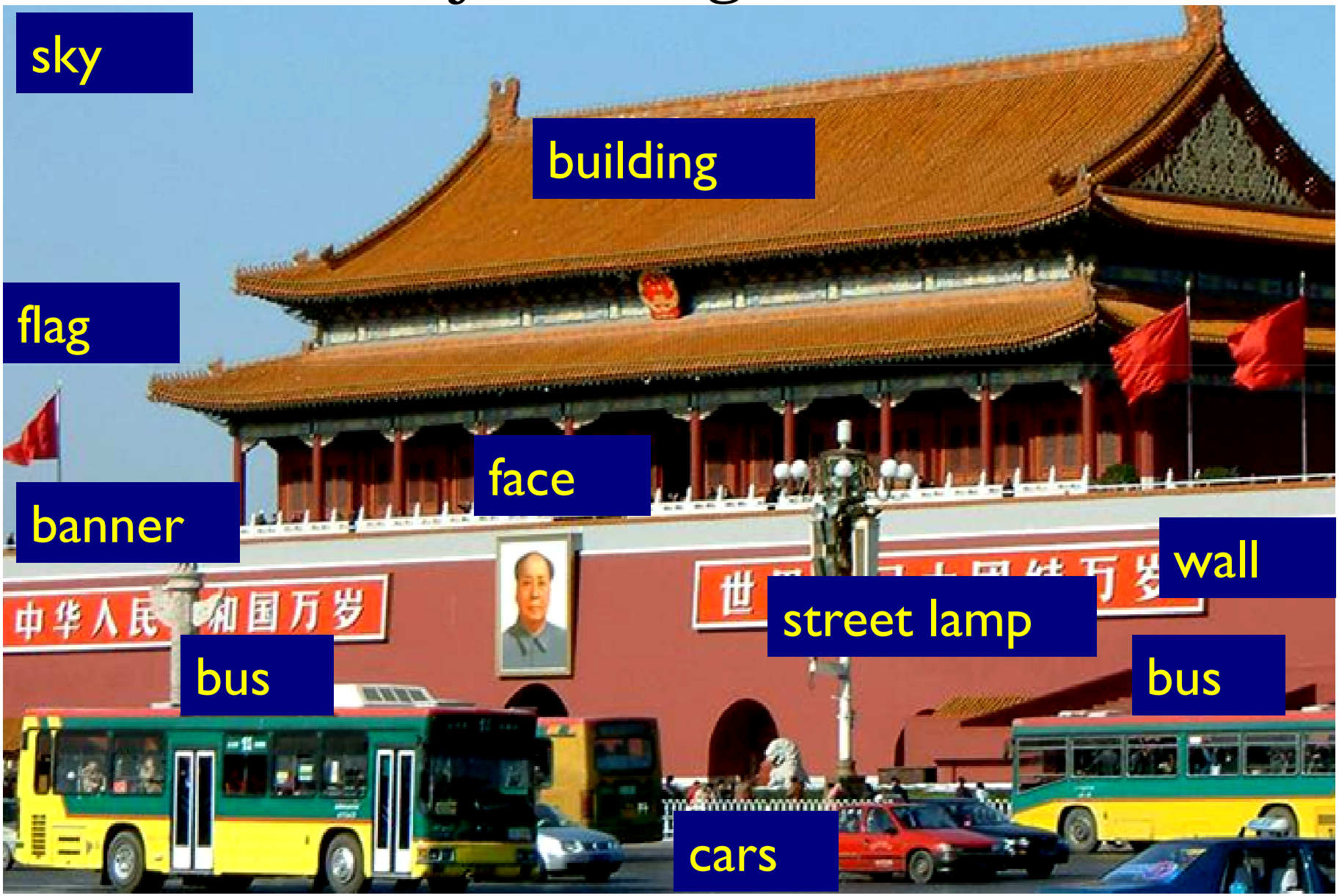




Goesele et al.

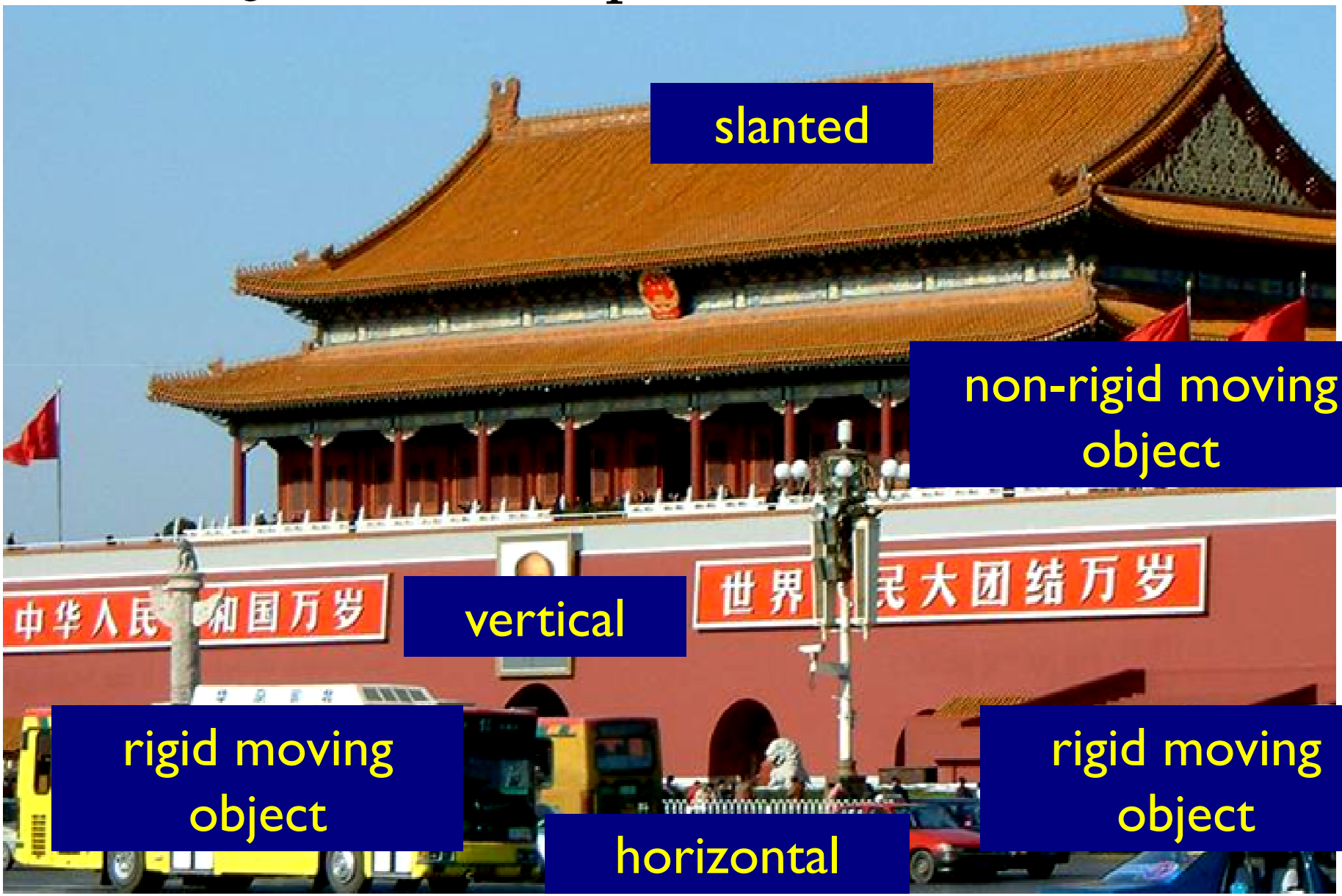
Vision as a source of semantic information

Object categorization



Scene and context categorization

Qualitative spatial information



Challenges: viewpoint variation

Michelangelo 1475-1564

Challenges: illumination

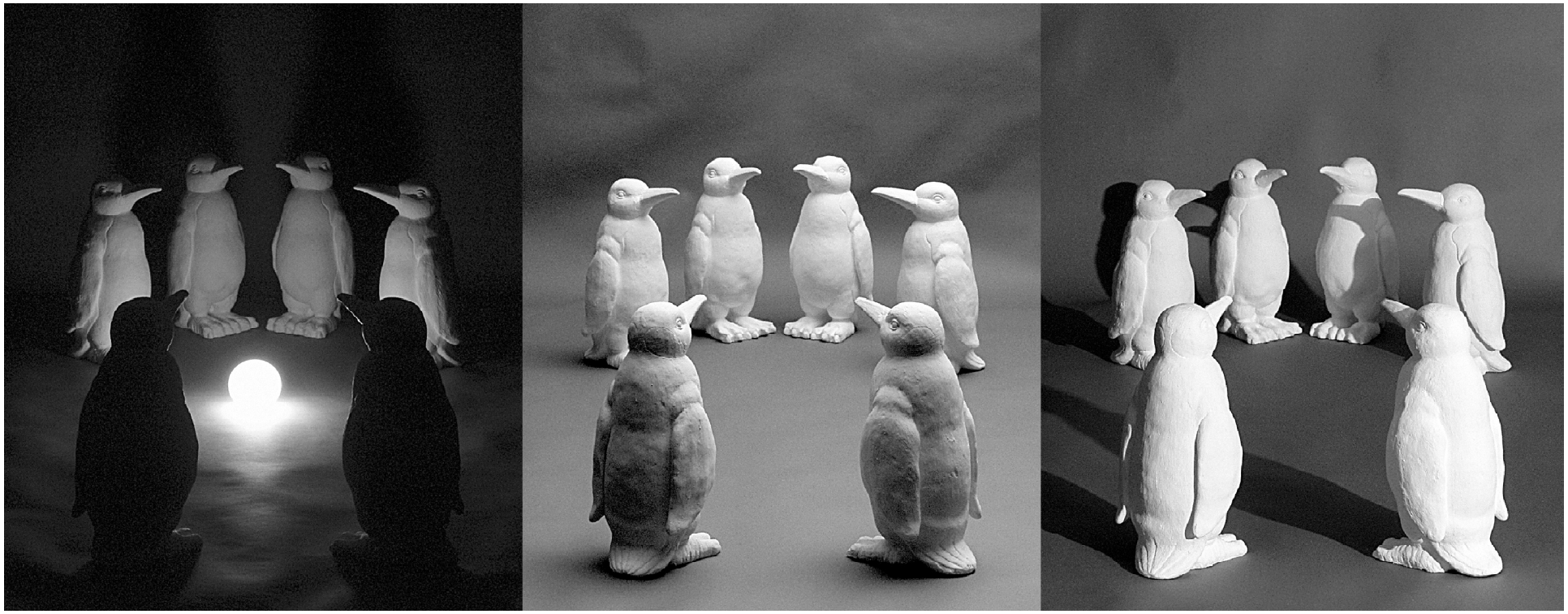
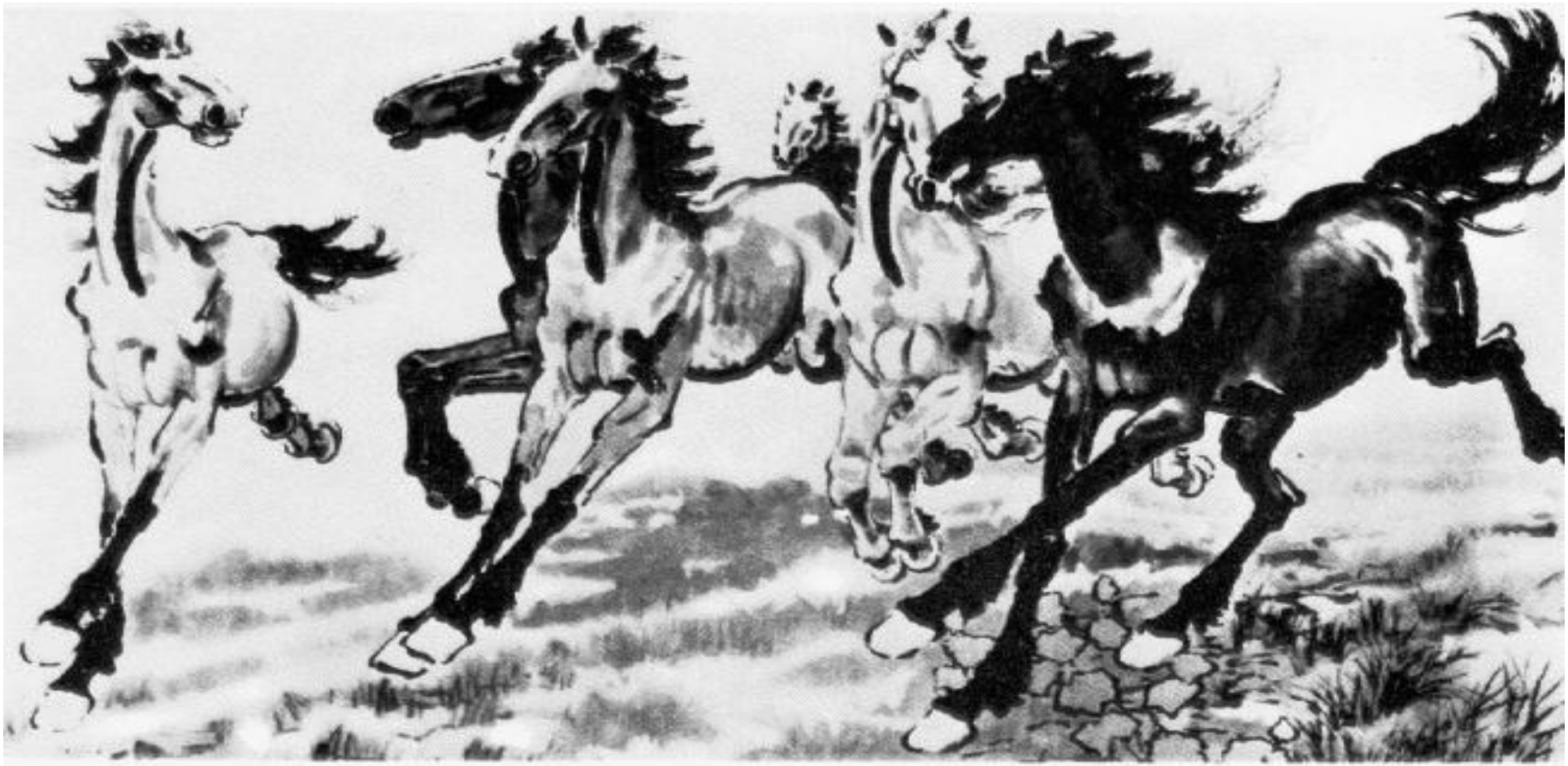


image credit: J. Koenderink

Challenges: scale

slide credit: Fei-Fei, Fergus & Torralba

Challenges: deformations



Xu, Beihong 1943

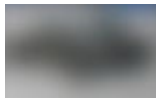
Challenges: occlusions

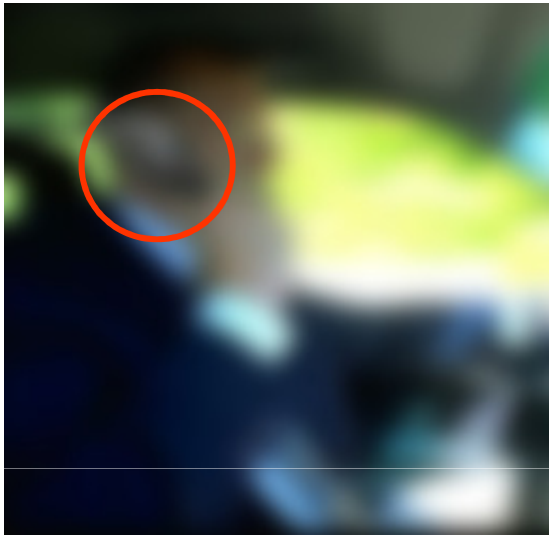
Magritte, 1957

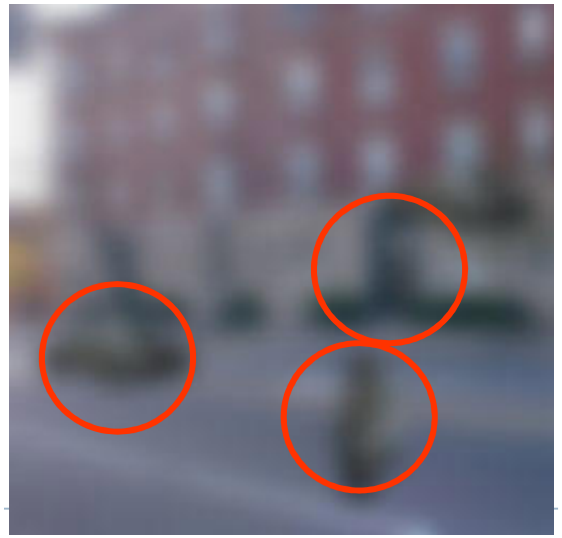
Challenges: background clutter

Challenges: object intra-class variation

Challenges: local ambiguity





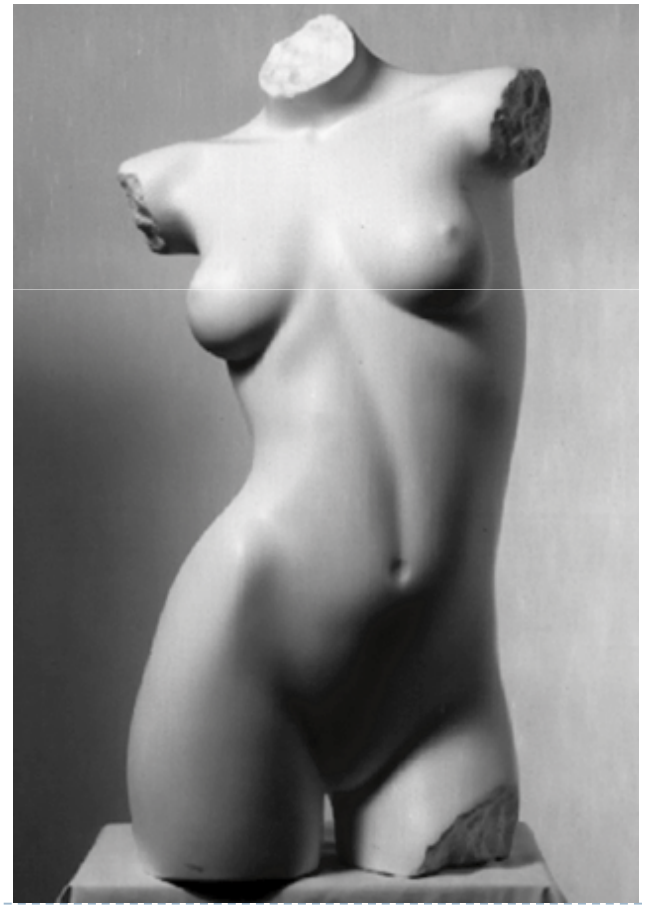


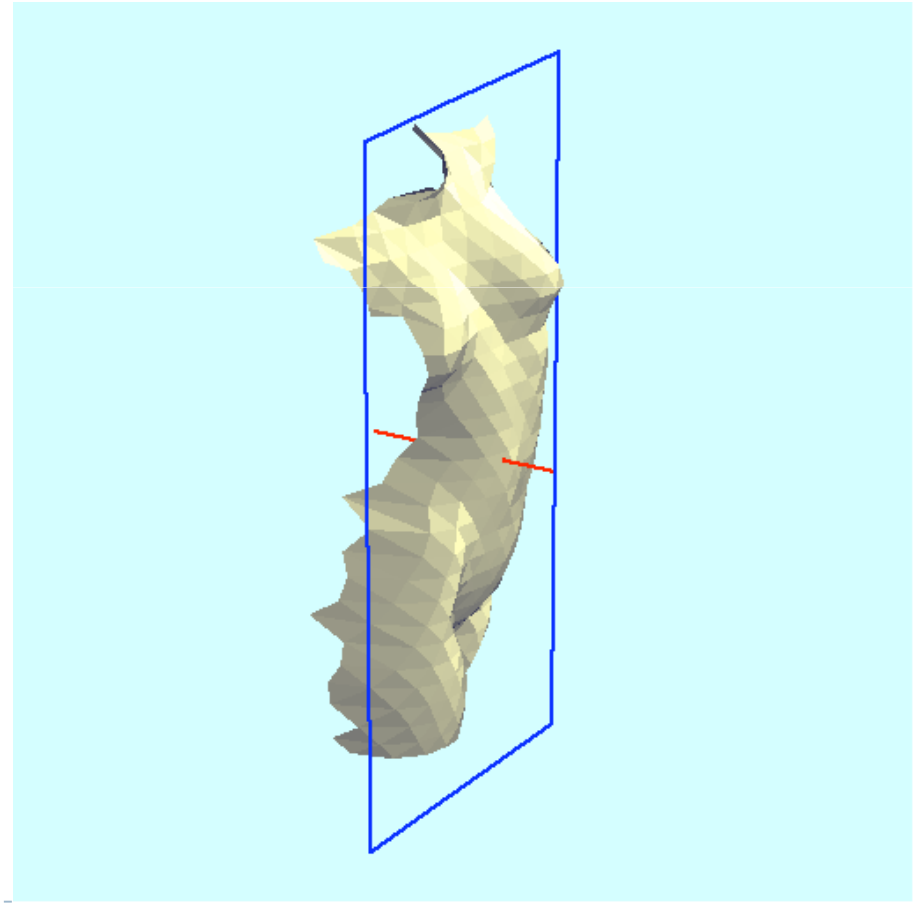
Challenges or opportunities?

- Images are confusing, but they also reveal the structure of the world through numerous cues
- Our job is to interpret the cues!
- i.e. Linear perspective, texture gradient,…

Challenges or opportunities?

- Shape and lighting cues: Shading



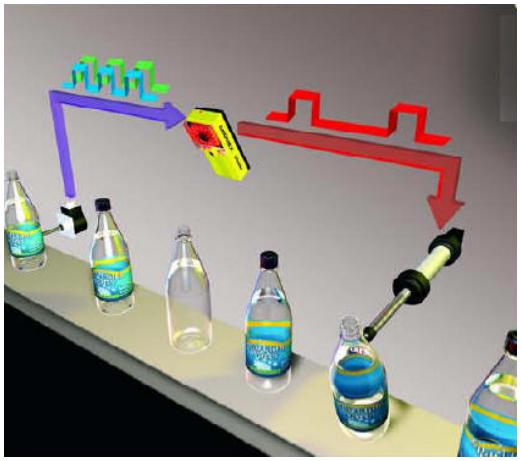


Challenges or opportunities?

- Grouping cues: Similarity

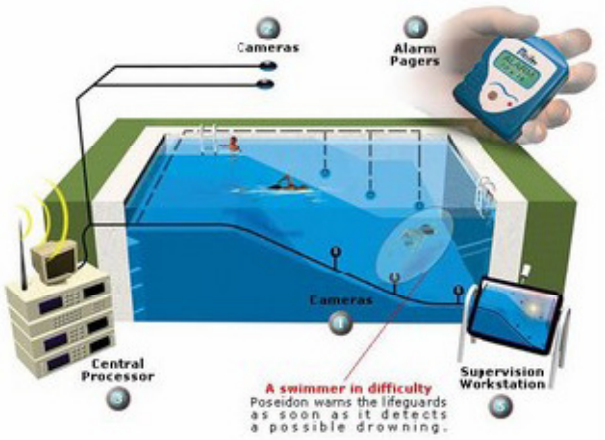
(color, texture, proximity, shape…)

Applications



Factory inspection

Reading license plates, checks, ZIP codes



Monitoring for safety (Poseidon)

Surveillance

Autonomous driving, robot navigation

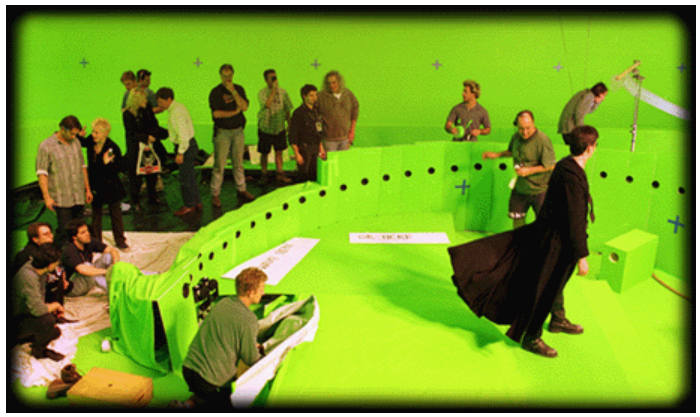
Driver assistance

More info: http://people.cs.ubc.ca/~lowe/vision.html

Applications

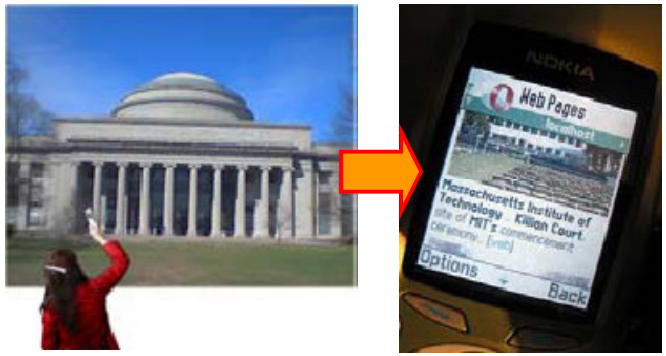
Assistive technologies

Entertainment
(Sony EyeToy)



Movie special effects

Digital cameras (face detection for setting focus, exposure)



Visual search
(MSR Lincoln)

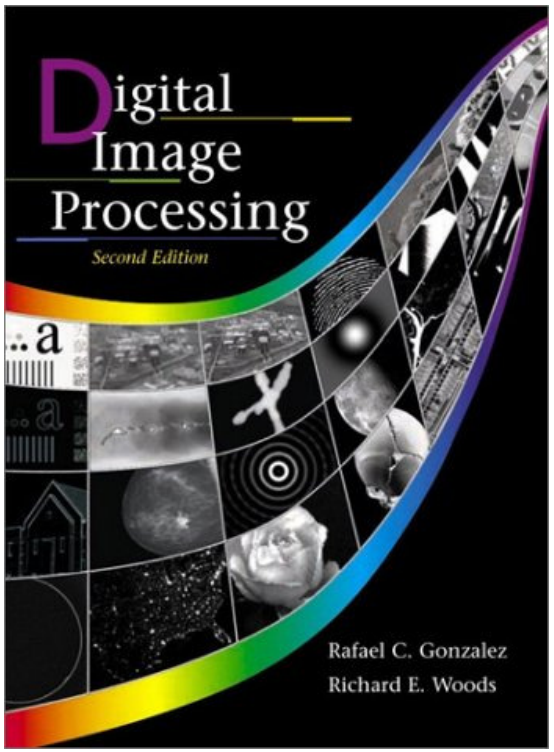
More info: http://people.cs.ubc.ca/~lowe/vision.html

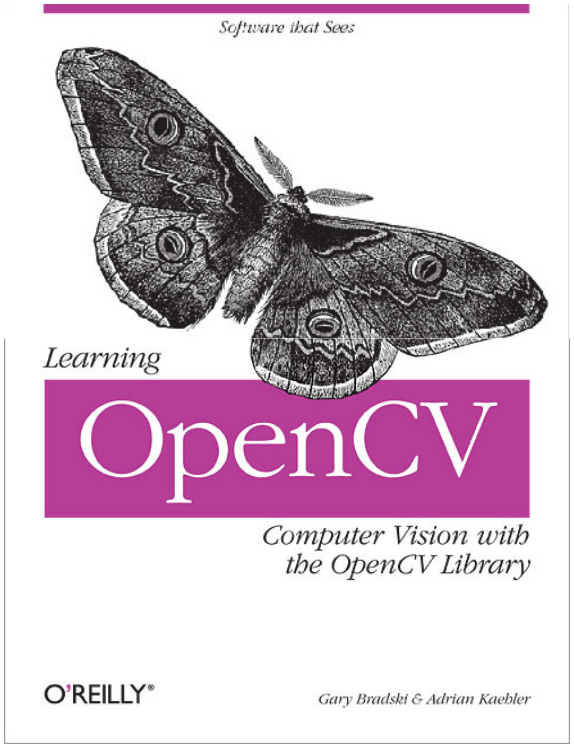
Applications

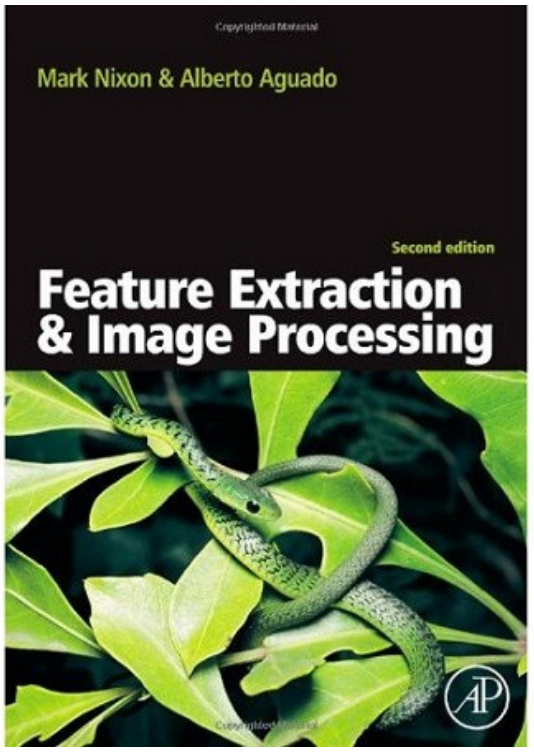
- A famous application by using not only cameras…

Applications

Litterature

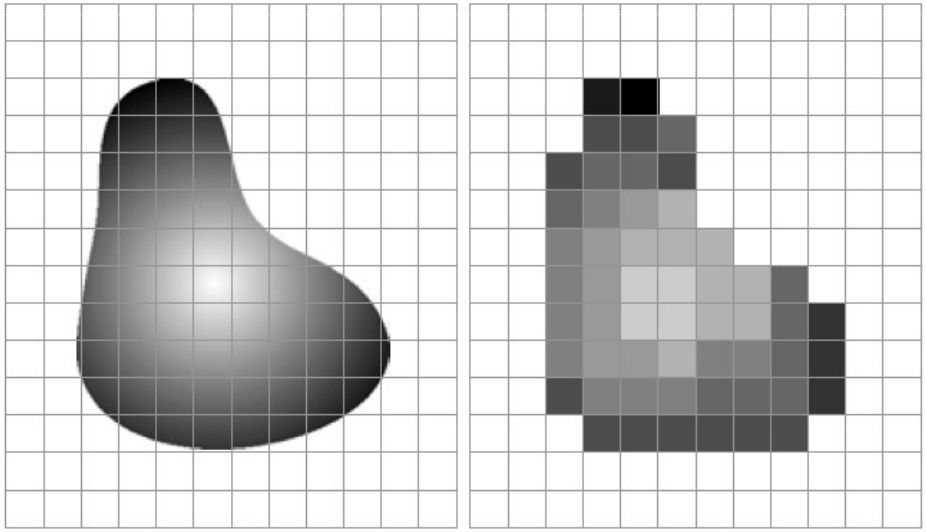






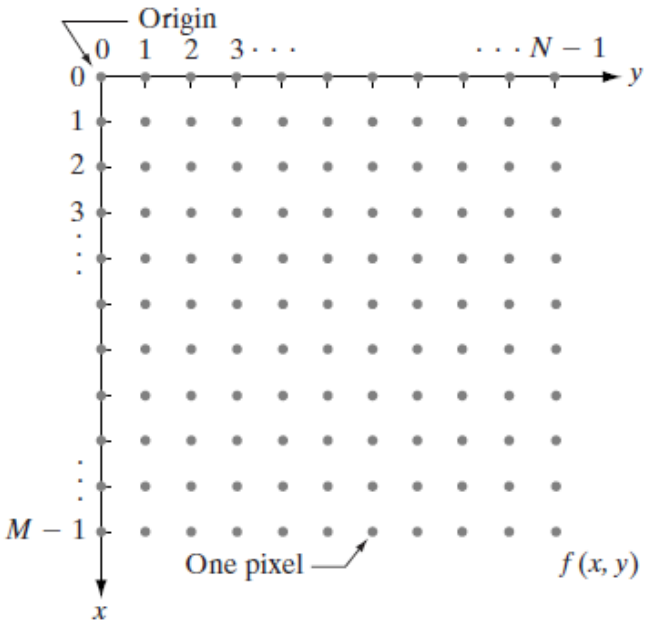
Digital images and processing

- An **image** may be defined as a two-dimensional function, f(x, y), where x and y are spatial (plane) coordinates, and the amplitude of f at any pair of coordinates (x, y) is called the intensity or gray level of the image at that point.
- When x, y, and the amplitude values of f are all finite, discrete quantities, we call the image a **digital image**.



Digital images and processing

- The field of digital **image processing** refers to processing digital images by means of a digital computer.
- A digital image is composed of a finite number of elements, each of which has a particular location and value. These elements are referred to as picture elements, image elements, pels, and **pixels**.
- Coordinate convention to represent digital images:



Digital images and processing

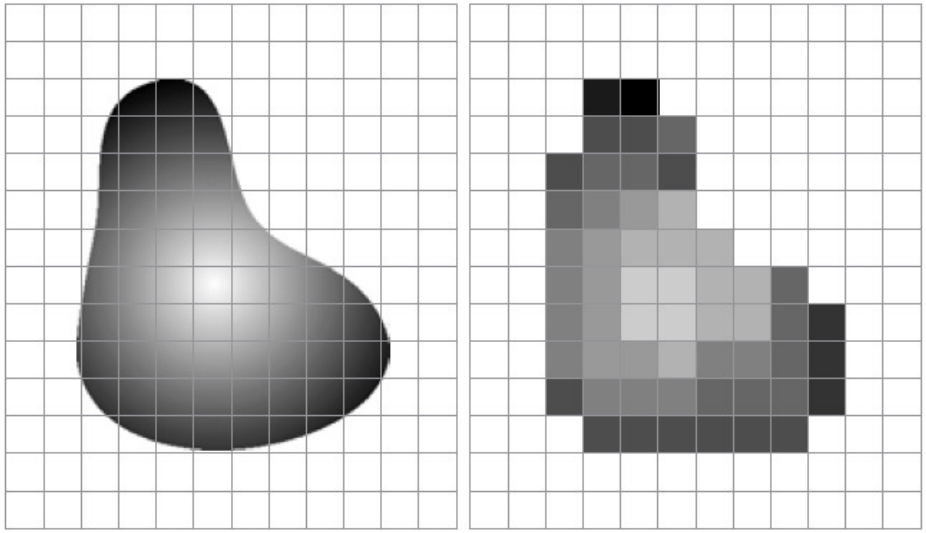
- Sampling and Quantization

An image may be continuous with respect to the x- and y-coordinates, and also in amplitude.

To convert it to digital form, we have to sample the function in both coordinates and in amplitude.

Digitizing the coordinate values is called **sampling**.

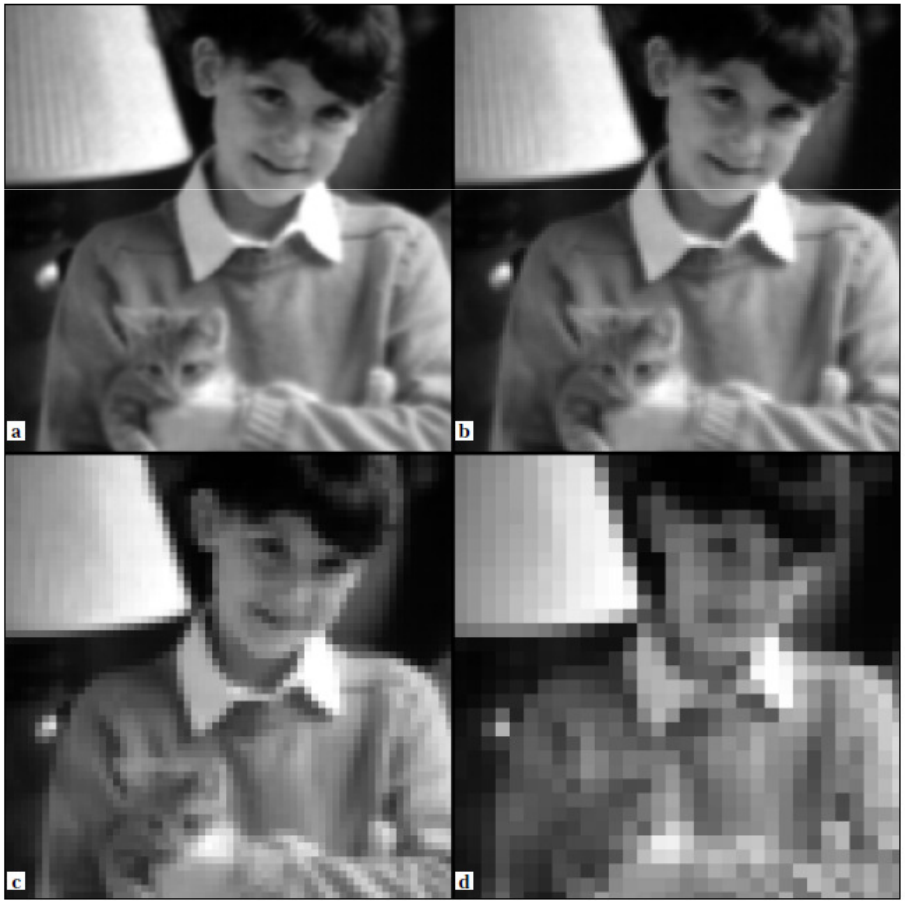
Digitizing the amplitude values is called **quantization**.



Digital images and processing

- Spatial and Gray-Level Resolution

Spatial resolution is the smallest discernible detail in an image: the smallest number of discernible line pairs per unit distance; for example, 100 line pairs per millimeter.



Four representations of the same image, with variation in the number of pixels used:

a) 256x 256;
b) 128 x128;
c) 64x 64;
d) 32 x 32.

Digital images and processing

- Spatial and Gray-Level Resolution

Gray-level resolution refers to the smallest discernible change in gray level.

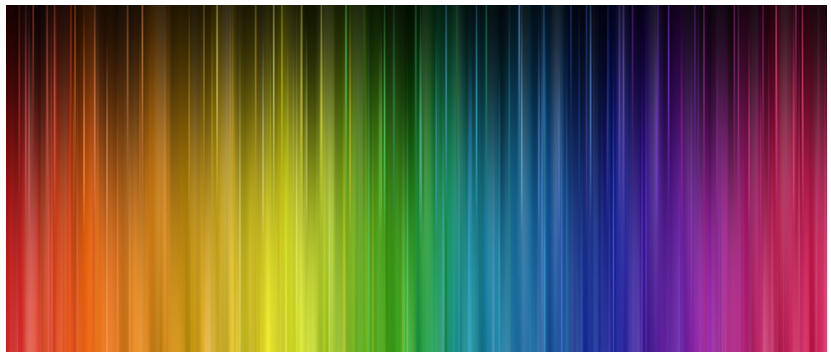
Four representations of the same image, with variation in the number of grey levels used:

a) 32;
b) 16;
c) 8;
d) 4.

Digital images and processing

- Color imaging

Most real-world images are not monochrome, of course, but full color!



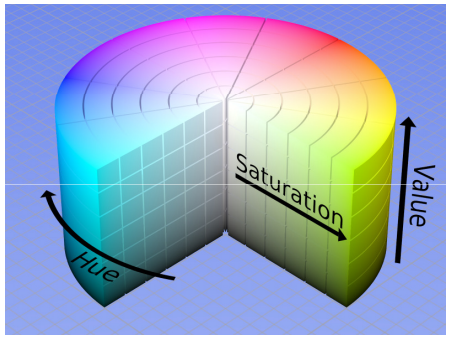
- Images from digital cameras are usually in Red Green Blue (RGB) colorspace since according to the three color theory of Thomas Young, all the colors are perceived by our visual system as linear combination of the basic colors.
- A number of color spaces or color models have been suggested and each one of them has a specific color coordinate system and each point in the color space represents only one specific color
- Each color model may be useful for specific applications. For our purposes…

Digital images and processing

- HSV colorspace

Cylindrical-coordinate representations of points in an RGB color model, which rearrange the geometry of RGB in an attempt to be more **perceptually relevant** than the cartesian representation.

HSV stands for **hue**, **saturation**, and **value**:



- Hue represents color. It is an angle from 0 degrees to 360 degrees.
- Saturation indicates the range of grey in the color space. It ranges from 0 to 100%. When the value is '0,' the color is grey and when the value is '1,' the color is a primary color.
- Value is the brightness of the color and varies with color saturation. It ranges from 0 to 100%. When the value is '0' the color space will be totally black. With the increase in the value, the color space brightness up and shows various colors.

Digital images and processing

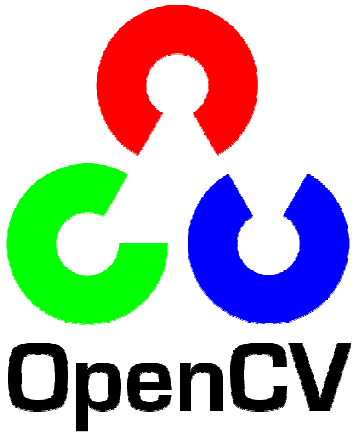
- HSV colorspace

Example: skin detection using HSV colorspace

Thresholds: H(0-20); S (30-150); V(80-255)

What is OpenCV?

- OpenCV is an open source computer vision library available from http://SourceForge.net/projects/opencvlibrary
- The library is written in C and C++ and runs under Linux, Windows and Mac OS X.
- There is active development on interfaces for Python, Ruby, Matlab, and other languages.
- One of OpenCV's goals is to provide a simple-to-use computer vision infrastructure that helps people build fairly sophisticated vision applications quickly.



What is OpenCV?

- The OpenCV library contains over 500 functions that span many areas in vision, including:
  - factory product inspection,
  - medical imaging,
  - security,
  - user interface,
  - camera calibration,
  - stereo vision,
  - robotics.

- There is a Yahoo groups forum where users can post questions and discussion at *http://groups.yahoo.com/group/OpenCV; it has about 20,000 members.*

Let's start!

1. Install OpenCV library

   *OpenCV-2.1.0-win32-vs2008.exe (*Version 2.1 is more stable…)

   to a folder (without any spaces in it), say "C:\OpenCV2.1\".

   We'll refer to this path as $openCVDir

   During installation, enable the option "Add OpenCV to the system PATH for all users".

2. Open Visual Studio 2008 and configure it

Visual Studio 2008 configuration

- Tools > Options > Projects and Solutions > VC++ Directories
- Choose "Show directories for: Include files"
  - Add "$openCVDir\include\opencv“
- Choose "Show directories for: Library files"
  - Add "$openCVDir\lib“
- Choose "Show directories for: Source files"
  - Add "$openCVDir\src\cv"
  - Add "$openCVDir\src\cvaux"
  - Add "$openCVDir\src\cxcore"
  - Add "$openCVDir\src\highgui”

Visual Studio 2008 Project dependencies

3. Create a new project WIN32 and add the OpenCV dependencies

- Open Project Properties: Project > %projectName% Properties...
- Open Linker Input properties: Configuration Properties > Linker > Input
- Open the "..." window to edit "Additional Dependencies" and on each line put:
  - "cv210.lib"
  - "cxcore210.lib"
  - "highgui210.lib“
- And any other lib file necessary for your project

Our very first program

Try it! copy – paste – build - run – see

```
#include "highgui.h“

int main()
{
        char* path=" "start.jpg";
        IplImage* img = cvLoadImage( path );
        cvNamedWindow( "Example1", CV_WINDOW_AUTOSIZE);
        cvShowImage( "Example1", img );
        cvWaitKey(0);
        cvReleaseImage( &img );
        cvDestroyWindow( "Example1" );
}
```

Cue: put the image in the same directory!

Documentation

Documentation can be found in the .../opencv/docs subdirectory.

Moreover, the OpenCV Wiki is available on the Web:

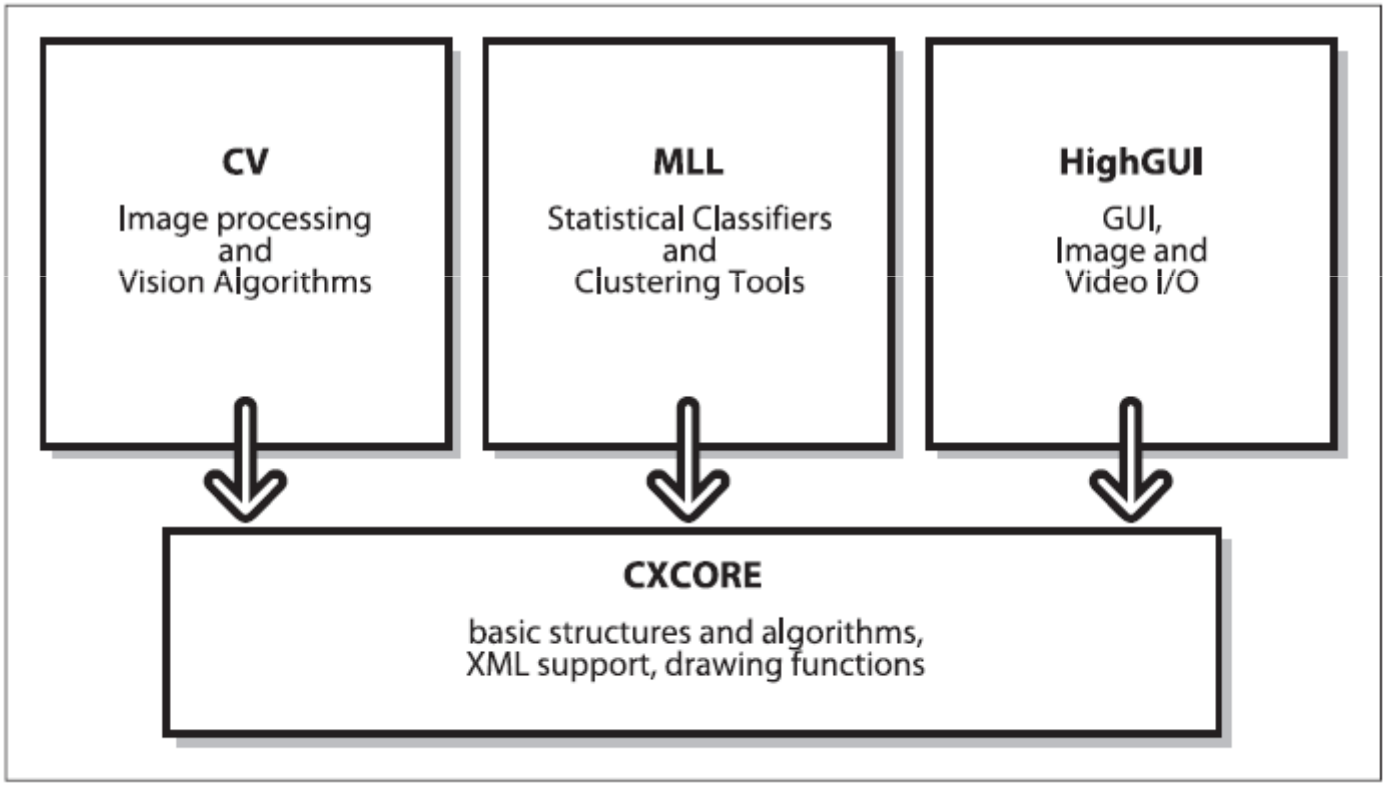
http://opencv.willowgarage.com/documentation/

and it's structured as follow:

- **CXCORE**: Contains data structures, matrix algebra, data transforms, object persistence, memory management, error handling, and dynamic loading of code as well as drawing, text and basic math.
- **CV**: Contains image processing, image structure analysis, motion and tracking, pattern recognition, and camera calibration.
- Machine Learning (**MLL**): Contains many clustering, classification and data analysis functions.
- **HighGUI**: Contains user interface GUI and image/video storage and recall
- **CVCAM**: Camera interface.

OpenCV Structure and Content

OpenCV is broadly structured into 5 main components,

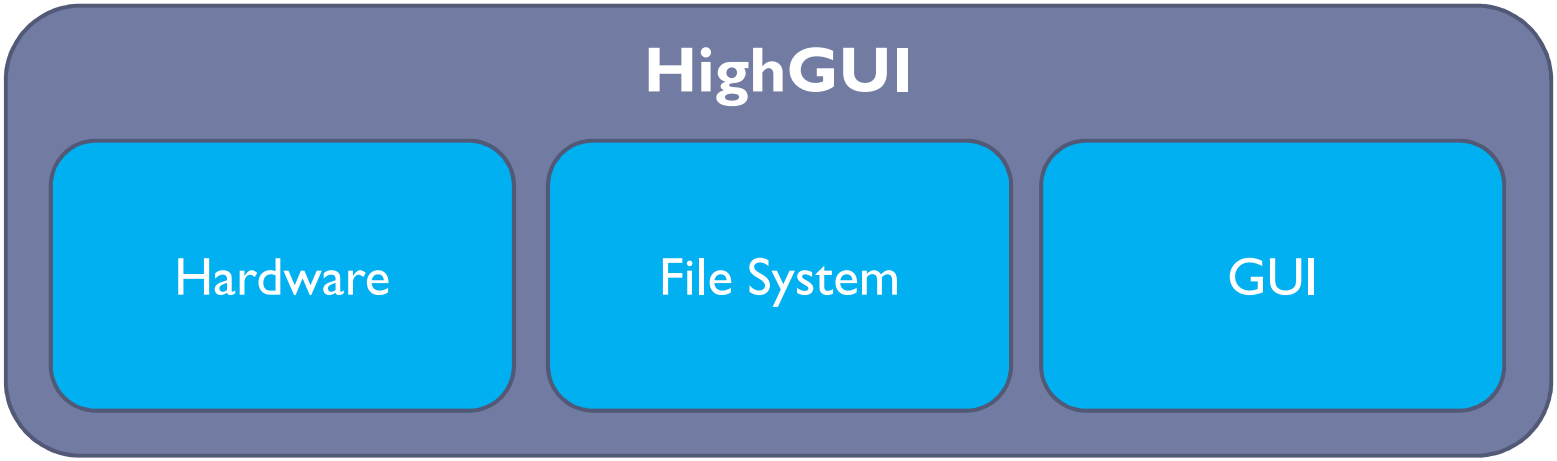


OpenCV Structure and Content

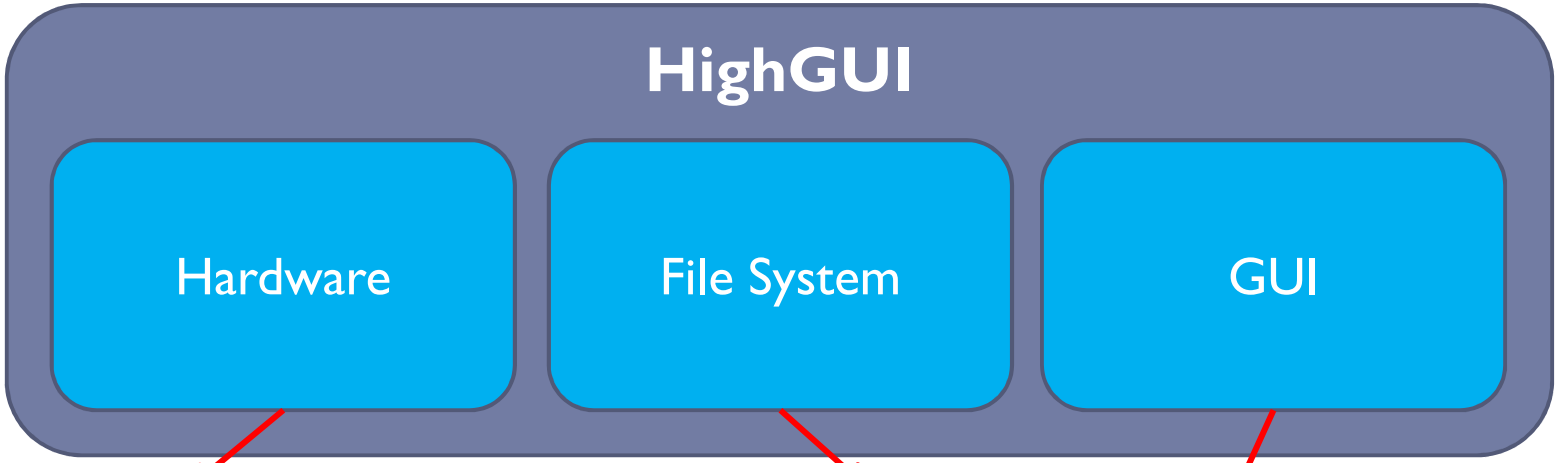
HighGUI allows us to interact with the operating system, the file system, and hardware such as cameras.

With HighGUI we can quite easily open windows; display images; read and write graphics-related files (both images and video); handle simple mouse, pointer, and keyboard events.; create useful stuff like sliders and then add them to windows…

It can be divided into 3 parts:



OpenCV Structure and Content



operation of cameras: in most operating systems, interaction with a camera is a tedious and painful task. HighGUI allows an easy way to query a camera and retrieve the latest image from the camera. It hides all of the nasty stuff , and that keeps us happy.

loading and saving images

functions that will allow us to open a *window* and throw an image into that window. It also allows us to register and respond to **mouse and keyboard events** on that window.

CXCORE: Primitive Data Types

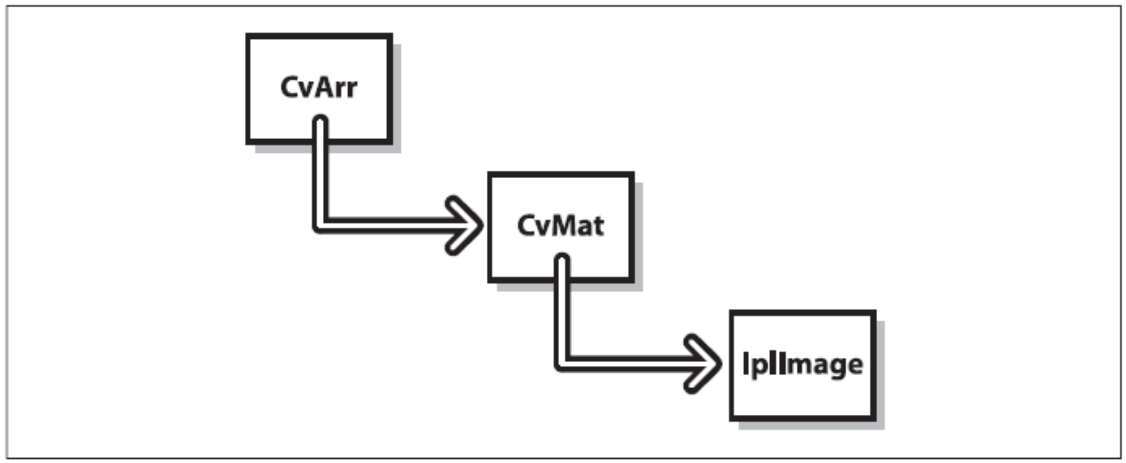
OpenCV data types are not primitive from the point of view of C, but they are all simple structures, and we will regard them as atomic:

- **CvScalar**: container for 1-,2-,3- or 4-tuples of doubles.
- **CvPoint**: (x,y) integer points in image
  - CvPoint2D32f: (x,y) float points in $\Re^2$
  - CvPoint3D32f (x,y,z) float points in $\Re^3$
- **CvSize**: (width, height) integer size of image
- **CvRect**: (x, y, width, height) integer portion of image (rectangle)
  Example: a white rectangle between pixels (5, 10) and (100, 100);

```
cvRectangle(
    myImg,
    cvPoint(5,10),
    cvPoint(100,100),
    cvScalar(255,255,255)
);
```

Images and matrices

- When using OpenCV, you will repeatedly encounter the **IplImage** data type.
- IplImage is the basic structure used to encode what we generally call "images".
- Images may be grayscale, color, four-channel (RGB+alpha), and each channel may contain any of several types of integer or floating point numbers.
- **IplImage** is derived from the matrix type (**CvMat**), which is derived from the array type (**CvArr**):



So, let's see what CvMat is like

Matrices

CvMat

- there is no "vector" construct in OpenCV. Whenever we want a vector, we just use a matrix with one column (or one row, if we want a transpose or conjugate vector).

Routine that creates a new 2D matrix:

```
cvMat* cvCreateMat ( int rows, int cols, int type );
```

- type can be any of a long list of predefined types of the form:

  CV_<bit_depth>(S|U|F)C<number_of_channels>

  CV_32FC1: 32-bit floats;

  CV_8UC3: unsigned integer 8-bit 3-channels

Matrices

- Create and allocate a matrix
  ```
  CvMat* cvCreateMat(int rows, int cols, int type)
  void cvCreateData(CvMat* mat);
  ```
- Clone a matrix
  ```
  mat_new=cvCloneMat(mat);
  ```
- Accessing matrix data
  ```
  elemtype CV_MAT_ELEM(CvMat*, elemtype, row, col)
  ```
- Clean up
  ```
  cvReleaseMat(&mat);
  ```

Example:

```
CvMat* mat = cvCreateMat( 5, 5, CV_32FC1 );
float element_3_2 = CV_MAT_ELEM( *mat, float, 3, 2 );
```

Images

`IplImage` is a `CvMat` with some extra goodies buried in it to make the matrix interpretable as an image.

```
typedef struct _IplImage
{
    int  nSize;
    int  ID;
    int  nChannels;
    int  alphaChannel;
    int  depth;
    char colorModel[4];
    char channelSeq[4];
    int  dataOrder;
    int  origin;
    int  align;
    int  width;
    int  height;
    struct _IplROI *roi;
    struct _IplImage *maskROI;
    void  *imageId;
    struct _IplTileInfo *tileInfo;
    int  imageSize;
    char *imageData;
    int  widthStep;
    int  BorderMode[4];
    int  BorderConst[4];
    char *imageDataOrigin;
}
IplImage;
```

RELEVANT ONES

nSize: sizeof(IplImage)
nChannels: Number of channels
Depth: Pixel depth in bits. The supported depths are:
- IPL_DEPTH_8U - Unsigned 8-bit integer
- IPL_DEPTH_8S - Signed 8-bit integer
- IPL_DEPTH_16U - Unsigned 16-bit integer
- IPL_DEPTH_16S - Signed 16-bit integer
- IPL_DEPTH_32S - Signed 32-bit integer
- IPL_DEPTH_32F - Single-precision floating point
- IPL_DEPTH_64F - Double-precision floating point

Origin: IPL_ORIGIN_TL or IPL_ORIGIN_BL
Width: image width in pixels
Height: Image height in pixels
Roi: Region Of Interest. If not NULL, only this region will be processed.
imageSize: Image data size in bytes
imageData: A pointer to the aligned image data
widthStep: The size of an aligned image row, in bytes
imageDataOrigin: A pointer to the origin of the image data

Images

- Allocate an image
  IplImage* cvCreateImage(CvSize size, int depth, int channels)

```
// Allocate a 1-channel byte image IplImage*
img1 = cvCreateImage(cvSize(640,480),IPL_DEPTH_8U,1);

// Allocate a 3-channel float image IplImage*
img2 = cvCreateImage(cvSize(640,480),IPL_DEPTH_32F,3);
```

- Load an image
  img=cvLoadImage(fileName);
- Clone an image
  img_new=cvCloneImage(img);
- Save an image
  cvSaveImage(fileName,img);
- Clean up
  cvReleaseImage(&img);

Images

- Accessing Image Data

There isn't an Opencv function to do that. We need to computes a pointer as the head of the relevant row and then move it to the relevant column by considering the widthStep.

Example: max out (saturate) only the "G" and "B" channels of an RGB image

```
void saturate( IplImage* img ) {
    for( int y=0; y<img->height; y++ ) {
        uchar* ptr = (uchar*) (
            img->imageData + y * img->widthStep
        );
        for( int x=0; x<img->width; x++ ) {
            ptr[3*x+1] = 255;
            ptr[3*x+2] = 255;
        }
    }
}
```

RGB image

The pointer **prt** scans each row

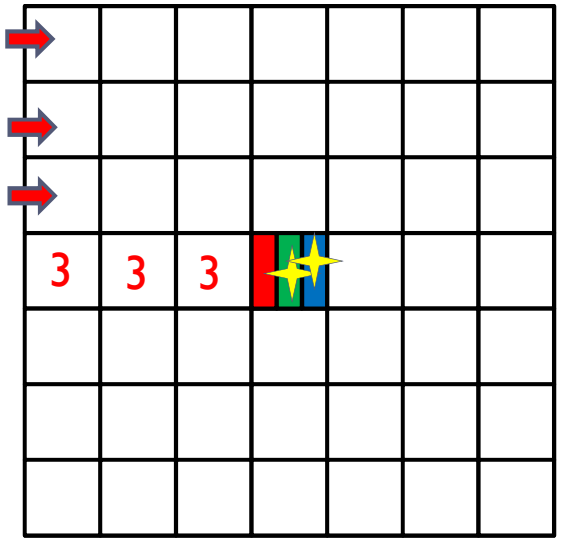
widthStep contains the number of bytes between points in the same column and successive rows

Because this is a 3-channel image, the location of channel c is 3x+c

Images

```
void saturate( IplImage* img ) {
    for( int y=0; y<img->height; y++ ) {
        uchar* ptr = (uchar*) (
            img->imageData + y * img->widthStep
        );
        for( int x=0; x<img->width; x++ ) {
            ptr[3*x+1] = 255;
            ptr[3*x+2] = 255;
        }
    }
}
```

Y=0
Y=1
Y=2
Y=3



i.e.
Change channel green and blue of pixel (3,3)

Cool! I'm on the correct row

But there're 3 channels…

HIGHGUI: Windows

As we're working with images, we need to display them. The Highgui section of Opencv includes some routines for managing windows and displaying images.

- Create a window
  `int cvNamedWindow(const char* name, int flags)`
  name is the name of the window; suggested flag: `CV_WINDOW_AUTOSIZE`

- Move a window
  `void cvMoveWindow(const char* name, int x, int y)`
  name is the name of the window; x,y is the offset from the UL corner of the screen

- Display an image
  `void cvShowImage(const char* name, const CvArr* image)`
  name is the name of the window; image is the image to be shown

- Close a window and clean up
  `void cvDestroyWindow(const char* name)`
  `void cvDestroyWindow(const char* name)` or
  `void cvDestroyAllWindows(void)`

Example 1

- Now we can understand the test-program we run previously…
- **Load an image from disk and displays it on the screen**

```
#include "highgui.h"

int main()
{
    char* path="images.jpg";
    IplImage* img = cvLoadImage( path );

    cvNamedWindow( "Example1", CV_WINDOW_AUTOSIZE );
    cvShowImage( "Example1", img );
    cvWaitKey(0);
    cvReleaseImage( &img );
    cvDestroyWindow( "Example1" );
}
```

WaitKey

- Waitkey causes OpenCV to wait for a specified number of milliseconds for a user keystroke. If the key is pressed within the allotted time, the function returns the key pressed; otherwise, it returns 0.

Example: tell OpenCV to wait 100 ms for a key stroke

```
while( 1 ) {
      if( cvWaitKey(100)==27 ) break;
}
```

ASCII value 27 is the Escape key

Another example:

```
cvWaitKey(0)
```

It will wait indefinitely until a keystroke is received and then return that key.

HIGHGUI: Videos

Playing a video with OpenCV is almost as easy as displaying a single picture.
`CvCapture` is the video capturing structure.

- Initializing capture from a file
  `CvCapture* cvCaptureFromFile(const char* filename)`

- Initializing capture from a camera
  `CvCapture* cvCaptureFromCAM(0)`
  0 value grabs frames from the default camera

- Grab and return a frame from a camera or file
  `IplImage* cvQueryFrame(CvCapture*  cap)`

- Get capture device properties
  `double cvGetCaptureProperty(CvCapture*  cap, int property_id)`

Videos

- Get capture device properties
  `double cvGetCaptureProperty(CvCapture*  cap, int property_id)`

Property identifier. Can be one of the following:

CV_CAP_PROP_POS_MSEC: Film current position in milliseconds or video capture timestamp
CV_CAP_PROP_POS_FRAMES - 0-based index of the frame to be decoded/captured next
CV_CAP_PROP_POS_AVI_RATIO - Relative position of the video file (0 - start of the film, 1 - end of the film)
CV_CAP_PROP_FRAME_WIDTH - Width of the frames in the video stream
CV_CAP_PROP_FRAME_HEIGHT - Height of the frames in the video stream
CV_CAP_PROP_FPS - Frame rate
CV_CAP_PROP_FOURCC - 4-character code of codec
CV_CAP_PROP_FRAME_COUNT - Number of frames in the video file
CV_CAP_PROP_BRIGHTNESS - Brightness of the image (only for cameras)
CV_CAP_PROP_CONTRAST - Contrast of the image (only for cameras)
CV_CAP_PROP_SATURATION - Saturation of the image (only for cameras)
CV_CAP_PROP_HUE - Hue of the image (only for cameras)

Videos

- Release the capture source
  void cvReleaseCapture(CvCapture capture)**

- Save a frame into a video file
  CvVideoWriter* cvCreateVideoWriter(const char* filename, int compression, double fps, CvSize size, int is_color)

  int cvWriteFrame(CvVideoWriter* wrter, const IplImage* image)

Example:

```
CvVideoWriter *writer = 0;
int isColor = 1;
int fps = 25;
int frameW = 640;
int frameH = 480
writer=cvCreateVideoWriter("out.avi", -1, fps, cvSize(frameW,frameH), isColor);
```

- Releasing the video writer
  void cvReleaseVideoWriter(CvVideoWriter writer)**

Example 2

- Create a OpenCV program that **reads an video from disk and displays it on the screen**

```
#include "highgui.h"

int main()
{
    char* path="testavi.avi";
    cvNamedWindow( "Example2", CV_WINDOW_AUTOSIZE );
    CvCapture* capture = cvCreateFileCapture( path );
    IplImage* frame;
    while(1) {
        frame = cvQueryFrame( capture );
        if( !frame ) break;
        cvShowImage( "Example2", frame );
        char c = cvWaitKey(33);
        if( c == 27 ) break;
    }
    cvReleaseCapture( &capture );
    cvDestroyWindow( "Example2" );
}
```

Example 2

```
#include "highgui.h"

int main()
{
    char* path="testavi.avi";
    cvNamedWindow( "Example2", CV_WINDOW_AUTOSIZE );
    CvCapture* capture = cvCreateFileCapture( path );
    IplImage* frame;
    while(1) {
        frame = cvQueryFrame( capture );
        if( !frame ) break;
        cvShowImage( "Example2", frame );
        char c = cvWaitKey(33);
        if( c == 27 ) break;
    }
    cvReleaseCapture( &capture );
    cvDestroyWindow( "Example2" );
}
```

Once we have displayed the frame, we then wait for (1/30) 33 ms (if the frame rate is 30 fps) If the user hits the Esc key (ASCII 27), then we will exit the read loop. Otherwise, 33 ms will pass and we will just execute the loop again.

Sidebars

As we're working with videos, we can add useful graphic tools, like sidebars or trackbars..

- This function create a trackbar and attach it to the specified window

```
int cvCreateTrackbar(const char* trackbarName,
const char* windowName, int* value, int count,
CvTrackbarCallback onChange)
```

Parameters:

`trackbarName` – Name of the created trackbar.

`windowName` – Name of the window.

`value` – Pointer to an integer variable, whose value will reflect the position of the slider. Upon creation, the slider position is defined by this variable.

`count` – Maximal position of the slider. Minimal position is always 0.

`onChange` – Pointer to the function to be called every time the slider changes position. This function should be prototyped as `void Foo(int);` Can be NULL if callback is not required.

Example 3

- Create a OpenCV program that reads an video from disk and displays it on the screen and add **a sidebar showing the frame number**

```
#include "cv.h"
#include "highgui.h"

int g_slider_position = 0;
CvCapture* g_capture = NULL;

void onTrackbarSlide(int pos) {
    cvSetCaptureProperty(
        g_capture,
        CV_CAP_PROP_POS_FRAMES,
        pos
    );
}
```

Example 3

```
int main()
{
    char* path="testavi.avi";

    cvNamedWindow( "Example3", CV_WINDOW_AUTOSIZE );
    CvCapture* g_capture = cvCreateFileCapture( path );

    int frames = (int) cvGetCaptureProperty(
        g_capture,
        CV_CAP_PROP_FRAME_COUNT
    );

    if( frames!= 0 ) {
        cvCreateTrackbar(
        "Frame",
        "Example3",
        &g_slider_position,
        frames,
        onTrackbarSlide
        );
    }
    IplImage* frame;
```

Example 3

```
while(1) {
    frame = cvQueryFrame( g_capture );
    if( !frame ) break;
    cvShowImage( "Example3", frame );

    cvCreateTrackbar(
        "Frame",
        "Example3",
        &g_slider_position,
        frames,
        onTrackbarSlide
        );


    char c = cvWaitKey(33);
    if( c == 27 ) break;

    ++g_slider_position;
}
cvReleaseCapture( &g_capture );
cvDestroyWindow( "Example3" );
```

A simple image transformation

We saw that images can be represented with different colorspaces.
OpenCV allows to convert an image from one color space to another with the function:

```
void cvCvtColor(const CvArr* src, CvArr* dst, int code)
```

Parameters:

`src` – The source 8-bit (8u), 16-bit (16u) or single-precision floating-point (32f) image

`dst` – The destination image of the same data type as the source. The number of channels may be different

`code` – Color conversion operation, i.e.

- CV_RGB2GRAY
- CV_GRAY2RGB
- CV_RGB2HSV
- CV_HSV2RGB
- …

Example 4

- Create a OpenCV program that **reads a RGB video from disk, displays it and save it in gray levels**.

```
#include "cv.h"
#include "highgui.h"

CvCapture* g_capture = NULL;

int main()
{
    char* path="testavi.avi";
    char* path2="testavi_write.avi";

    cvNamedWindow( "Example4", CV_WINDOW_AUTOSIZE );
    cvNamedWindow( "Example4gray", CV_WINDOW_AUTOSIZE );

    CvCapture* capture = cvCreateFileCapture( path );

    IplImage* frame;
    IplImage* gray;

    double fps = cvGetCaptureProperty (
        capture,
        CV_CAP_PROP_FPS
    );
```

Example 4

```
CvSize size = cvSize(
    (int)cvGetCaptureProperty( capture, CV_CAP_PROP_FRAME_WIDTH),
    (int)cvGetCaptureProperty( capture, CV_CAP_PROP_FRAME_HEIGHT)
);

CvVideoWriter* writer=0;

writer = cvCreateVideoWriter(
    path2,
    -1,
    fps,
    size,
    0
);
```

Example 4

```
while(1) {
    frame = cvQueryFrame( capture );
    if( !frame ) break;

    cvShowImage( "Example4", frame );

    gray = cvCreateImage( cvSize(frame->width, frame->height), IPL_DEPTH_8U, 1 );
    cvCvtColor( frame, gray, CV_BGR2GRAY );

    char c = cvWaitKey(33);

    cvShowImage( "Example4gray", gray );
    cvWriteFrame( writer, gray );

    if( c == 27 ) break;
}

    cvReleaseVideoWriter( &writer );
    cvReleaseCapture( &capture );
    cvDestroyWindow( "Example4" );
    cvDestroyWindow( "Example4gray" );
    return(0);
}
```

Practice 1/1

Write a program which:

- reads a RGB image;
- shows the original image;
- shows the R,G,B channels on 3 different images;
- transform the the image in HSV colorspace;
- shows the H,S,V channels on 3 different images;
- save each channel separately

Cue: see cvSplit

Practice 1/2

Write a program which:

- gathers frames from a webcam
- shows the original frame;
- transform the frame in gray-level colorspace;
- shows the gray frame on a different window
- (save the gray video for 5 seconds)

Cue: use

```
CvCapture* capture = cvCaptureFromCAM(0);
```

Instead of `cvCaptureFromFile`

Resize

- We often encounter an image of some size that we would like to convert to an image of some other size. We may want to upsize (zoom in) or downsize (zoom out) the image.
- Zooming requires two steps:
  1. the creation of new pixel locations;
  2. the assignment of gray levels to those new locations.

- Suppose that we have an image of size 128*128 pixels and we want to enlarge it 8 times to 1024*1024 pixels. How can we assign intensity values to the "new" pixels?

  The easiest way is called *nearest neighbor interpolation*: we look for the closest pixel in the original image and assign its gray level to the new pixel in the grid.

  Although nearest neighbor interpolation is fast, it has the undesirable feature that it produces a checkerboard effect that is particularly objectionable at high factors of zooming.

Resize

- Example:

  Images zoomed from 128*128, 64*64, and 32*32 pixels to 1024*1024 pixels, using nearest neighbour gray-level interpolation

Resize

- A slightly more sophisticated way of accomplishing gray-level assignments is *bilinear interpolation* (linear interpolation of 2 variables) using the four nearest neighbors of a point.

Resize

- **OpenCV**

```
void cvResize(
    const CvArr*    src,
    CvArr*          dst,
    int             interpolation = CV_INTER_LINEAR
);
```

EXAMPLE

```
img2 = cvCreateImage(cvSize(img->width/3,img->height/3), img->depth,img->nChannels);
cvResize(img, img2, CV_INTER_LINEAR);
```

Interpolation	Meaning
CV_INTER_NN	Nearest neighbor
CV_INTER_LINEAR	Bilinear

Practice 1/3

Write a program which:

- reads a RGB image;
- shows the original image;
- changes image size (zooming in and out) with both the nearest and the bilinear interpolation methods;
- shows the new images;
- save all images

Which sizes and methods are acceptable for you?

Region Of Interest (ROI)

Regions of Interest have a great practical importance, since in many situations they speed up computer vision operations by allowing the code to process only a small subregion of the image.

Given a rectangular subregion of interest in the form of a **`CvRect`**, you may pass an image pointer and the rectangle to **`cvSetImageROI()`** to "turn on" ROI; "turn off " ROI by passing the image pointer to **`cvResetImageROI().`**

Example: we want to load an image and modify a region of that image.

1. read an image;
2. set the x, y, width, and height of the intended ROI;
3. increment all of the pixels in the region with a specific value;
4. release the ROI with `cvResetImageROI()`

Example 5

- Create a OpenCV program that **reads an image, convert it in gray levels and add 150 to the gray levels of a Region Of Interest with**
  x = 50
  y = 50
  width = 100
  height = 200

Example 5

```
#include "cv.h"
#include "highgui.h"

int main()
{
    char* path="Umbria.jpg";

    cvNamedWindow( "Example5", CV_WINDOW_AUTOSIZE );
    cvNamedWindow( "Example5ROI", CV_WINDOW_AUTOSIZE );
    cvNamedWindow( "Example5Gray", CV_WINDOW_AUTOSIZE );

    IplImage* image;
    IplImage* image_new;
    IplImage* gray;

    int x = 50;
    int y = 50;
    int width = 100;
    int height = 200;
    int add = 150;

    image = cvLoadImage( path );
    cvShowImage( "Example5", image );

    gray = cvCreateImage( cvSize(image->width, image->height), IPL_DEPTH_8U, 1 );
    cvCvtColor( image, gray, CV_BGR2GRAY );
```

Example 5

```
    cvShowImage( "Example5Gray", gray );

    image_new = cvCloneImage (gray);
    cvSetImageROI(image_new, cvRect(x,y,width,height));
    cvAddS(image_new, cvScalar(add),image_new);

    cvResetImageROI(image_new);
    cvShowImage( "Example5ROI", image_new );

    cvWaitKey();
    cvReleaseImage(&image);
    cvReleaseImage(&image_new);
    cvDestroyWindow( "Example5" );
    cvDestroyWindow( "Example5ROI" );
    cvDestroyWindow( "Example5Gray" );
    return(0);
}
```

Practice 1/4

Write a program which:

- reads a RGB image;
- shows the original image;
- sets a ROI;
- shows the RGB ROI on a new window;
- transform the whole RGB image in gray-level colorspace;
- saturates ROI (255);
- shows the new ROI on a new window;
- shows the new image with the white ROI on a new window;
- saves all images

Drawing

Something that frequently occurs is the need to draw some kind of picture or to draw something on top of an image obtained from somewhere else.

- Draw a line segment connecting two points

```
void cvLine(CvArr* img, CvPoint pt1, CvPoint pt2,
CvScalar color, int thickness=1, int lineType=8, int shift=0)
```

- Draw a rectangle

```
void cvRectangle(CvArr* img, CvPoint pt1, CvPoint pt2,
CvScalar color, int thickness=1, int lineType=8, int shift=0)
```

- Draw a circle

```
void cvCircle(CvArr* img, CvPoint center, int radius,
CvScalar color, int thickness=1, int lineType=8, int shift=0)
```

- Draw an ellipse

```
void cvEllipse(CvArr* img, CvPoint center, CvSize axes,
double angle, double start_angle, double end_angle,
CvScalar color, int thickness=1, int lineType=8, int shift=0)
```

Writing

OpenCV has one main routine, that just throws some text onto an image. The text indicated by text is printed with its lower-left corner of the text box at origin and in the color indicated by color.

- Initialize font structure

```
void cvInitFont(CvFont* font, int fontFace, double hscale,
double vscale, double shear=0, int thickness=1,
int lineType=8)
```

- Draw a text string.

```
void cvPutText(CvArr* img, const char* text, CvPoint org,
const CvFont* font, CvScalar color)
```

Drawing and Writing

Example: draw a red rectangle corresponding to the ROI and write the blue text “My ROI” close to the rectangle .

```
cvRectangle( image,  cvPoint(x,y), cvPoint(x+width,y+height), CV_RGB(255, 0, 0), 4);

CvFont font;
double hScale=1.0;
double vScale=1.0;
int    lineWidth=3;
cvInitFont(&font,CV_FONT_HERSHEY_SIMPLEX|CV_FONT_ITALIC, hScale,vScale,0,lineWidth);
cvPutText (image, "My ROI", cvPoint(x+width,y+height), &font, CV_RGB(0, 0, 255));
```

Mouse management

A typical callback mechanism is to enable response to mouse clicks.

We must first write a callback routine that OpenCV can call whenever a mouse event occurs.

- Create a Callback
  `void CvMouseCallback(int event, int x, int y, int flags,void* param)`

Event	Numerical value
CV_EVENT_MOUSEMOVE	0
CV_EVENT_LBUTTONDOWN	1
CV_EVENT_RBUTTONDOWN	2
CV_EVENT_MBUTTONDOWN	3
CV_EVENT_LBUTTONUP	4
CV_EVENT_RBUTTONUP	5
CV_EVENT_MBUTTONUP	6
CV_EVENT_LBUTTONDBLCLK	7
CV_EVENT_RBUTTONDBLCLK	8
CV_EVENT_MBUTTONDBLCLK	9

Mouse management

- Create a Callback
  `void CvMouseCallback(int event, int x, int y, int flags,void* param)`

x and y - coordinates of the mouse event

param - void pointer that can be used to have OpenCV pass in any additional information in the form of a pointer to whatever kind of structure you need

Flag	Numerical value
CV_EVENT_FLAG_LBUTTON	1
CV_EVENT_FLAG_RBUTTON	2
CV_EVENT_FLAG_MBUTTON	4
CV_EVENT_FLAG_CTRLKEY	8
CV_EVENT_FLAG_SHIFTKEY	16
CV_EVENT_FLAG_ALTKEY	32

Mouse management

- Register the callback

```
void cvSetMouseCallback( const char* window_name,
CvMouseCallback on_mouse, void* param = NULL)
```

`window_name:` name of the window to which the callback will be attached

`on_mouse:` the callback function

`param:` allows us to specify the *param* information that should be given to the callback whenever it is executed. This is the same *param* we were just discussing in regard to the callback prototype.

Example 6

- Create a OpenCV program that allows to **use a mouse to draw boxes on the screen**

```
#include "cv.h"
#include "highgui.h"

// Define our callback which we will install for mouse events.
void my_mouse_callback(
    int event,
    int x,
    int y,
    int flags,
    void* param
);

CvRect box;
bool drawing_box = false;

// A litte subroutine to draw a box onto an image
void draw_box( IplImage* img, CvRect rect ) {
    cvRectangle (
        img,
        cvPoint(box.x,box.y),
        cvPoint(box.x+box.width,box.y+box.height),  CV_RGB(255, 0, 0), 3
    );
}
```

Example 6

```
int main()
{
    char* path="Umbria.jpg";
    box = cvRect(-1,-1,0,0);
    cvNamedWindow( "Example6", CV_WINDOW_AUTOSIZE );
    IplImage* image;
    IplImage* temp;
    image = cvLoadImage( path );
    temp = cvCloneImage( image );

    // We install the callback -> 'param' is the image we are working with
    cvSetMouseCallback(
        "Example6",
        my_mouse_callback,
        (void*) image
    );

    while( 1 ) {
        cvCopyImage( image, temp );

        if( drawing_box )
            draw_box( temp, box );

        cvShowImage( "Example6", temp );
        if( cvWaitKey( 15 )==27 ) break;
    }
    cvReleaseImage(&image);
    cvDestroyWindow( "Example6" );
    return(0);
}
```

```
// Mouse callback   Left button -> we start a box
//                  Release the button -> add the box to the current image
//                  The mouse is dragged -> resize the box

void my_mouse_callback(
    int event, int x, int y, int flags, void* param
) {
    IplImage* image = (IplImage*) param;

    switch( event ) {
        case CV_EVENT_MOUSEMOVE: {
            if( drawing_box ) {
                box.width = x-box.x;
                box.height = y-box.y;
            }
        }
    break;
        case CV_EVENT_LBUTTONDOWN: {
            drawing_box = true;
            box = cvRect(x, y, 0, 0);
        }
    break;
        case CV_EVENT_LBUTTONUP: {
            drawing_box = false;
            if(box.width<0) {
                box.x+=box.width;
                box.width *=-1;
            }
            if(box.height<0) {
                box.y+=box.height;
                box.height*=-1;
            }
            draw_box(image, box);
        }
    break;
    }
}
```

HighGUI: no button! ☹

- Unfortunately, HighGUI does not provide any explicit support for buttons. We can easily use sliders (trackbars) that have only two positions (ON / OFF)
- Example 7:

```
#include "cv.h"
#include "highgui.h"

int g_switch_value = 0;

void switch_off_function(){ printf("OFF\n");}

void switch_on_function(){  printf("ON\n");}

void Switch_callback( int position ) {
    if( position == 0 ) {
        switch_off_function();
    } else {
    switch_on_function();
    }
}
```

Example 7: a trackbar to create a “switch”

```
int main() {

    cvNamedWindow( "ButtonWindow", 1 );

    // Create the trackbar, give it a name and tell it the name of the parent window.
    cvCreateTrackbar(
        "Switch",
        "ButtonWindow",
        &g_switch_value,
        1,
        Switch_callback
    );

    while( 1 ) {
    if( cvWaitKey(15)==27 ) break;
    }
}
```

Practice 1/5

Write a program which:

- loads a large image;
- reduces the image size (1/2);
- shows the image on a window;
- selects the ROI with the mouse clicks;
- shows a green rectangle corresponding to the selected ROI
- shows the ROI on a new window;
- writes a text box in the ROI saying ROI size "# x #', with the corresponding values;
- saves the images

Cue: see itoa

Practice 1/6

Write a program which:

- asks for the colorspace transformation;
- gathers frames from a webcam;
- prints the frame number on the image;
- changes the frame colorspace;
- shows one channel of the new colorspace on a new window with a trackbar indicating the time (in seconds);
- save the gray video for 5 seconds

See you after lunch