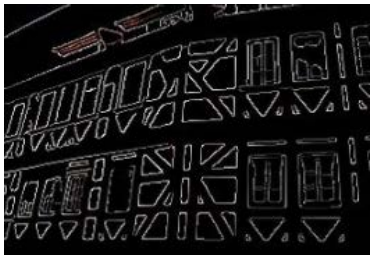
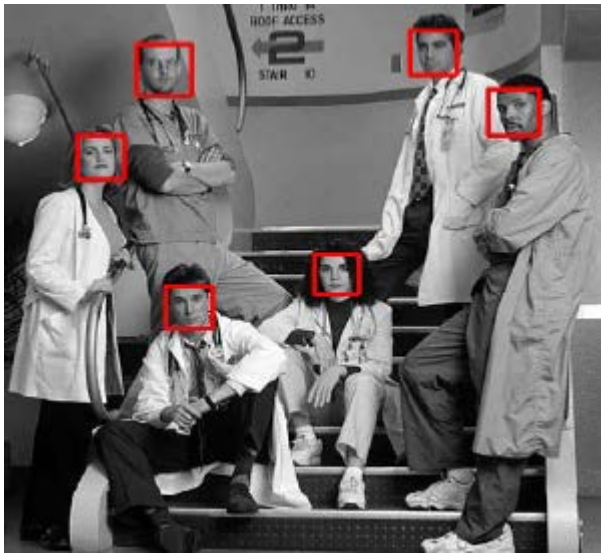
OpenCV

- Open Source Computer Vision Library (BSD-Lizenz) für C /C++ und Python
- wurde für PCs mit Intel-Architektur entwickelt und die Algorithmen wurden dafür optimiert
- gibt es für alle gängigen Betriebssysteme
- de-facto Standard für Bildverarbeitung und CV

Anwendungsgebiete

- Mustererkennung
- Objektidentifikation
- Objektsegmentation
- Objekterkennung
- Gesichtserkennung
- Gestenerkennung
- Motion Tracking
- 3D Rekonstruktion



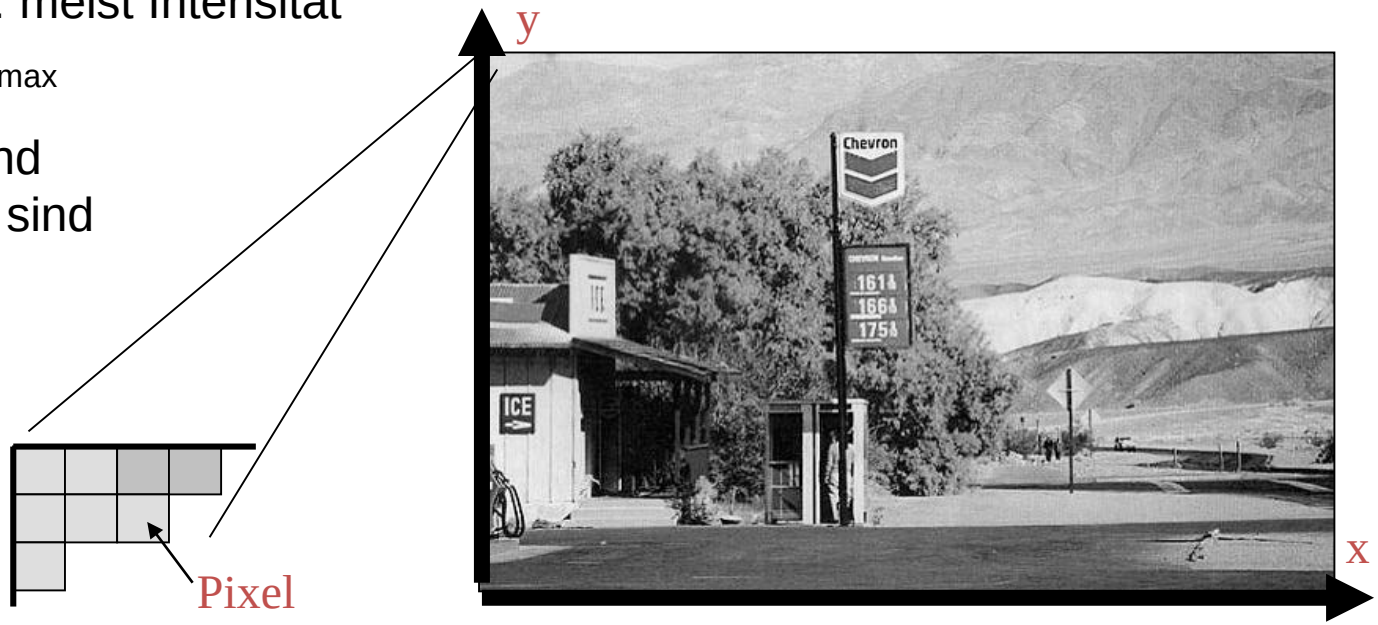


Wichtige Teile der Bibliothek

- **core** – Definition von Basisstrukturen und Basisfunktionen, z.B. arithmetische Operationen, Statistik
- **imgproc** – Bildverarbeitungsfunktionen, z.B. Filter
- **video** – Funktionen zur Videoanalyse, z.B. Bewegungsschätzung, Objekttracking
- **calib3d** – Basisfunktionen für Arbeit mit mehreren Kameras, z.B. Kamerakalibrierung
- **features2d** – Merkmalssberechnung
- **highgui** – Benutzerinterface, z.B. Bildanzeige, Bild- und Videoeinlesen
- **gpu** - GPU-basierte Algorithmen

Digitales Bild

- Das Bild besteht aus einer Menge von Bildelementen (***Pixel*** von „picture element“).
- Definitionsbereich: Ausdehnung in x- und y-Richtung, sowie Wellenlänge: $x_{min} \leq x < x_{max}$, $y_{min} \leq y < y_{max}$, $\lambda_{min} \leq \lambda < \lambda_{max}$.
- Wertebereich: meist Intensität $I_{min} \leq I(x,y,\lambda) < I_{max}$
- Definitions- und Wertebereich sind beschränkt.



Basisdatenstrukturen in C++

```
class CV_EXPORTS Mat
{public:
        // zahlreiche Methoden ...
  int flags; //enthält verschiedene Bit-Felder, z.B. Tiefe, Anzahl der Kanäle
  int dims; // für Felder mit Dimensionalität >= 2
  int rows, cols; // Anzahl der Zeilen und Spalten oder (-1, -1) für
  höherdimensionale Felder
  uchar* data; // Pointer auf die Daten
  int* refcount; // Pointer auf den Referenzzähler;
  // falls das Array auf Nutzer-allokierte Daten zeigt ist der Pointer NULL
  // andere Angaben ...
};
```

Arbeit mit Matrizen und Bildern in C++

```
Mat mtx(3, 3, CV_32F); // Erzeugen einer 3x3 Matrix vom Typ Float
Mat cmtx(10, 1, CV_64FC2); // Erzeugen eines komplexen Vektors
                           // mit 10 Elementen
Mat m = cv::imread("bild.bmp",0); //Einlesen eines Bildes
```

Datentypenbezeichnung in C++:

`CV_8U=0` (uchar), `CV_8S=1` (schar), `CV_16U=2` (ushort), `CV_16S=3` (short), `CV_32S=4` (int), `CV_32F=5` (float), `CV_64F=6` (double)

Mehrkanalige Typen:

`CV_8UC1 ... CV_64FC4` (für eine Anzahl von 1 bis 4 Kanälen)

OpenCV verwaltet den Speicher in C++ automatisch.

Speicher für Ausgabeparameter für Funktionen wird in C++ meist automatisch allokiert. Die Größe und der Typ wird automatisch anhand der Eingabeparameter bestimmt.

Vektoren und Matrizen haben Destruktoren zur Speicherfreigabe, wenn es nötig ist.

Beispielprogramm in C++

```
#include <opencv2\core\core.hpp>
#include <opencv2\highgui\highgui.hpp>

using namespace cv;

int main( int argc, char** argv )
{
  Mat bild;
  bild = imread("bild.bmp", 0);           // Einlesen des Bildes
  for(int i=0;i<bild.rows;i++)
     for(int j=0;j<bild.cols;j++)
       bild.at<uchar>(i,j) = 255 - bild.at<uchar>(i,j); //Invertieren
  imshow( "Test", bild );                 // Anzeige des Bildes
  waitKey(0);                             // Warten auf einen Tastendruck
  return 0;
}
```

Aufgabenstellung

1. Lesen Sie das Bild „zelle_grau.bmp“ aus dem Download-Bereich ein und zeigen Sie es auf dem Bildschirm an! (highgui)
2. Geben Sie die Bildgröße und die Anzahl der Farbkanäle aus! (core)
3. Bestimmen Sie den Durchschnittsgrauwert (mittleren Grauwert) im Bild. Wie stark weichen die einzelnen Werte von diesem Mittelwert ab? Eine Aussage darüber liefert die Standardabweichung. Geben Sie die beiden Werte aus! (core)
4. Bestimmen Sie sich den größten und den kleinsten Grauwert im Bild (core)
5. Ermitteln Sie, wie viele Pixel einen Grauwert besitzen, der größer als der mittlere Grauwert ist. Geben Sie diese Zahl aus und markieren Sie diese Pixel weiß im Bild! (core)
6. Erzeugen Sie aus dem Grauwertbild ein Farbbild und markieren Sie diese Pixel rot im Bild! (core)