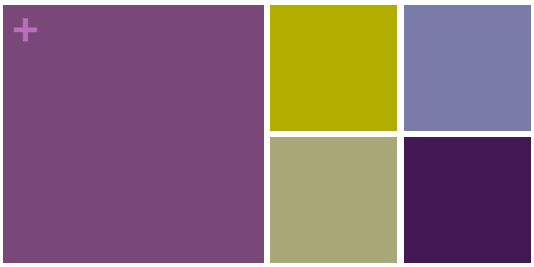

OpenCV

(Open Source Computer Vision)

Open Source Software

- What is it?
- License nuances: GNU GPL vs. GNU LGPL
  - Apache
  - BSD
  - MIT
  - Mozilla PL
  - Eclipse PL
- Free software is about freedom, not about money.

OpenCV History

- Officially launched in 1999 by Intel Research
- In the early days of OpenCV, the goals of the project were described as:
  - Advance vision research by providing an open basic vision infrastructure.
  - Disseminate vision knowledge.
  - Advance vision-based commercial applications.
- 1.0 version was released in 2006
- Version 2 in 2009 (We are using 2.4.1)
- Version 3 in 2015

Parts of OpenCV

- core
  - basic functionality
- imgproc
  - manipulation of images
- highgui
  - reading and writing images/videos

core

- Mat
  - The basic storage structure for images.

cout_mat

```
int main(int,char**)
{
    help();
    // Mat I = Mat::eye(4, 4, CV_64F);
    // I.at<double>(1,1) = CV_PI;
    // cout << "I = " << I << ";" << endl;

    int numRows = 10;
    int numColumns = 3;
    int numColors = 3;
    int numC2 = numColumns * numColors;
    Mat r = Mat(numRows, numColumns, CV_8UC3);
    // this randomly fills values
    randu(r, Scalar::all(0), Scalar::all(255));
```

```
// this fills the same values in order
uchar* data = r.data;
for(int i = 0; i < numRows; ++i) { // the rows
  for(int k = 0; k < numColumns; ++k) { // the columns
    for(int j = 0; j < numColors; ++j) { // the colors
      data[i*numC2+k*numColors + j] = (i+1); // set the row
    }
  }
}

Mat r2 = Mat(numRows, numColumns, CV_8UC3);
// this fills the same values in order
data = r2.data;
for(int i = 0; i < numRows; ++i) { // the rows
  for(int k = 0; k < numColumns; ++k) { // the columns
      data[i*numC2+k*numColors + 0] = (i+1); // set the row
      data[i*numC2+k*numColors + 1] = (k+1); // set the column
      data[i*numC2+k*numColors + 2] = 3; // set the color
  }
}
```

formatted python output.

```
r (python) = [[[1, 1, 1], [1, 1, 1], [1, 1, 1]],
  [[2, 2, 2], [2, 2, 2], [2, 2, 2]],
  [[3, 3, 3], [3, 3, 3], [3, 3, 3]],
  [[4, 4, 4], [4, 4, 4], [4, 4, 4]],
  [[5, 5, 5], [5, 5, 5], [5, 5, 5]],
  [[6, 6, 6], [6, 6, 6], [6, 6, 6]],
  [[7, 7, 7], [7, 7, 7], [7, 7, 7]],
  [[8, 8, 8], [8, 8, 8], [8, 8, 8]],
  [[9, 9, 9], [9, 9, 9], [9, 9, 9]],
  [[10, 10, 10], [10, 10, 10], [10, 10, 10]]];

r2 (python) = [[[1, 1, 3], [1, 2, 3], [1, 3, 3]],
  [[2, 1, 3], [2, 2, 3], [2, 3, 3]],
  [[3, 1, 3], [3, 2, 3], [3, 3, 3]],
  [[4, 1, 3], [4, 2, 3], [4, 3, 3]],
  [[5, 1, 3], [5, 2, 3], [5, 3, 3]],
  [[6, 1, 3], [6, 2, 3], [6, 3, 3]],
  [[7, 1, 3], [7, 2, 3], [7, 3, 3]],
  [[8, 1, 3], [8, 2, 3], [8, 3, 3]],
  [[9, 1, 3], [9, 2, 3], [9, 3, 3]],
  [[10, 1, 3], [10, 2, 3], [10, 3, 3]]];
```

Creating a Mat object explicitly

Mat() Constructor

```
Mat M(2,2, CV_8UC3, Scalar(0,0,255));
cout << "M = " << endl << " " << M << endl << endl;
```

```
M =
 [0, 0, 255, 0, 0, 255;
  0, 0, 255, 0, 0, 255]
```

For two dimensional and multichannel images we first define their size: row and column count wise.

Then we need to specify the data type to use for storing the elements and the number of channels per matrix point. To do this we have multiple definitions constructed according to the following convention:

CV_[The number of bits per item][Signed or Unsigned][Type Prefix]C[The channel number]

CV_8UC3 means we use unsigned char types that are 8 bit long and each pixel has three of these to form the three channels

Note

- You can fill out a matrix with random values using the randu() function. You need to give the lower and upper value for the random values:

```
Mat R = Mat(3, 2, CV_8UC3);
randu(R, Scalar::all(0), Scalar::all(255));
```

- Printing python formatted Matrices

```
cout << "R (python)  = " << endl <<
      format(R,"python") << endl << endl;
```

Creating a lookup table

```
int divideWith = 0; // convert our input string to number - C++ style
stringstream s;
s << argv[2];
s >> divideWith;
if (!s || !divideWith)
{
    cout << "Invalid number entered for dividing. " << endl;
    return -1;
}

uchar table[256];
for (int i = 0; i < 256; ++i)
   table[i] = (uchar)(divideWith * (i/divideWith));
```

DownSampling the image based on table

```
int i,j;
uchar* p;
for( i = 0; i < nRows; ++i)
{
    p = I.ptr<uchar>(i);
    for ( j = 0; j < nCols; ++j)
    {
        p[j] = table[p[j]];
    }
}
return I;
```

How the image matrix is stored in the memory

	Column 0	Column 1	Column ...	Column m
Row 0	0,0	0,1	...	0, m
Row 1	1,0	1,1	...	1, m
Row ...	...,0	...,1	...	..., m
Row n	n,0	n,1	n,....	n, m

For multichannel images the columns contain as many sub columns as the number of channels. For example in case of an RGB color system:

	Column 0			Column 1			Column ...			Column m		
Row 0	0,0	0,0	0,0	0,1	0,1	0,1	...	...	...	0, m	0, m	0, m
Row 1	1,0	1,0	1,0	1,1	1,1	1,1	...	...	...	1, m	1, m	1, m
Row ...	...,0	...,0	...,0	...,1	...,1	...,1	...	...	...	..., m	..., m	..., m
Row n	n,0	n,0	n,0	n,1	n,1	n,1	n,....	n,....	n,....	n, m	n, m	n, m

Mask operations on matrices

- Mask operations on matrices are quite simple.
  - recalculate each pixels value in an image according to a mask matrix (also known as kernel).
    - holds values that will adjust how much influence neighboring pixels (and the current pixel) have on the new pixel value.
  - From a mathematical point of view we make a weighted average, with our specified values.
- filter2D
  - Define mask Mat:

```
Mat kern = (Mat_<char>(3,3) <<  0, -1,  0,
                               -1,  5, -1,
                                0, -1,  0);
```

  - Call filter2d

```
filter2D(I, K, I.depth(), kern);
```

Linear Blending

- Problem: join 2 images
- Algorithm:
  - Preconditions: Images $X1_{r,c}$, $X2_{r,c}$, Blending Coefficient B in [0,1], offset *gamma*
  - Postcondition: Blended Image $Y1_{r,c}$
  - for each *i* in [0,*r*) and j in [0,*c*)
    - $Y1_{r,c} = B * X1_{r,c} + (1 - B) * X2_{r,c} + gamma$

OpenCV code for linear blending

We need two source images ($f_0(x)$ and $f_1(x)$). So, we load them in the usual way:

```
src1 = imread("../../images/LinuxLogo.jpg");
src2 = imread("../../images/WindowsLogo.jpg");
```

Warning: Since we are *adding src1* and *src2*, they both have to be of the same size (width and height) and type.

2. Now we need to generate the $g(x)$ image. For this, the function addWeighted comes quite handy:

```
beta = ( 1.0 - alpha );
addWeighted( src1, alpha, src2, beta, 0.0, dst);
```

Making cpp files that use OpenCV

```
all: cout_mat

cout_mat:
	g++ -Wall cout_mat.cpp -o cout_mat -l opencv_core

clean:
	rm -f cout_mat core
```

Our lab has the OpenCV library as a first class library (i.e. it is in the standard include and lib path. So all we need are the `-l` options. Aside from opencv_core, there is opencv_improc, opencv_highgui, and others.

Where do we go from here?

- OpenCV API - http://docs.opencv.org/2.4.11/modules/refman.html
- OpenCV User guide - http://docs.opencv.org/2.4.11/doc/user_guide/user_guide.html