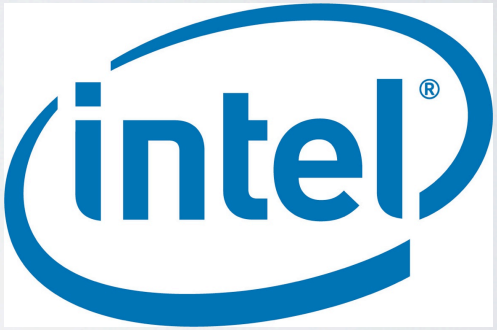
Introduction to OpenCV

Marvin Smith

Introduction

- OpenCV is an Image Processing library created by Intel and maintained by Willow Garage.
- Available for C, C++, and Python
- Newest update is version 2.2
- Open Source and free
- Easy to use and install



Installation Instructions

- For Mac OS X. Simply install Mac Ports then type
  sudo port install opencv

- Do not use synaptic on Linux to install OpenCV.
  It is version 1.2.

- For Linux and Windows, follow the installation guide at
  http://opencv.willowgarage.com/wiki/InstallGuide

- Linux users can come to me for help. I have built it on Ubuntu dozens of times. I have built it successfully on Windows once.

- Make sure to read the beginning as it gives you precise commands to install ffmpeg, libavformat-dev, libswscale-dev, and other required libraries.

- Follow instructions exactly!!!!!

BASIC OPENCV STRUCTURES

- **Point, Point2f** - 2D Point
- **Size** - 2D size structure
- **Rect** - 2D rectangle object
- **RotatedRect** - Rect object with angle
- **Mat** - image object

Point

- 2D Point Object
  - int x, y;
- Functions
  - Point.**dot**(<Point>) - computes dot product
  - Point.**inside**(<Rect>) - returns true if point is inside

```
Marvin-Smi
3, 3
2, 2
2.82843
A != B
B == C
Marvin-Smi
```

```
10 int main(int argc, char* argv[]){
11
12   Point a(1,1);
13   Point b(2,2);
14   Point c = a+b;
15
16   cout << c.x << ", " << c.y << endl;
17   c = a*2;
18   cout << c.x << ", " << c.y << endl;
19   cout << norm(b) << endl;
20
21   if(a==b)
22     cout << "A == B" << endl;
23   else
24     cout << "A != B" << endl;
25   if(b != c)
26     cout << "B != C" << endl;
27   else
28     cout << "B == C" << endl;
29   return 0;
30 }
```

- Math operators, you may use
  - **Point operator +**
  - **Point operator +=**
  - **Point operator -**
  - **Point operator -=**
  - Point operator *
  - Point operator *=
  - bool operator ==
  - bool operator != double norm

Size

- 2D Size Structure
  - int width, height;
- Functions
  - Point.area() - returns (width * height)

RECT

- 2D Rectangle Structure
- int x, y, width, height;
- Functions
- Point.tl() - return top left point
- Point.br() - return bottom right point

cv::Mat

- The primary data structure in OpenCV is the Mat object. It stores images and their components.

```
int main(int argc, char* argv[]){

  Mat image = imread(argv[1]);

  cout << "Colums = " << image.cols << endl;
  cout << "Rows   = " << image.rows << endl;
  cout << "Type   = ";

  if(image.type() == CV_8UC1)  cout << "CV_8UC1" << endl;
  else if(image.type() == CV_8UC3)  cout << "CV_8UC3" << endl;
  else if(image.type() == CV_32FC1) cout << "CV_32FC1" << endl;
  else if(image.type() == CV_32FC3) cout << "CV_32FC3" << endl;
  else cout << "Unknown" << endl;

  return 0;
```

- Main items
  - rows, cols - length and width(int)
  - channels - 1: grayscale, 3: BGR
  - depth: CV_<depth>C<num chan>
- See the manuals for more information

```
Marvin-Smiths-MacBook-Pro:Documents marvin_smith1$ g++ mat.cpp `pkg-config
Marvin-Smiths-MacBook-Pro:Documents marvin_smith1$ ./a.out photo.jpg
Colums = 400
Rows   = 300
Type   = CV_8UC3
Marvin-Smiths-MacBook-Pro:Documents marvin_smith1$
```

cv::Mat

- Functions
  - Mat.**at**<datatype>(row, col)[channel] - returns pointer to image location
  - Mat.**channels**() - returns the number of channels
  - Mat.**clone**() - returns a deep copy of the image
  - Mat.**create**(rows, cols, TYPE) - re-allocates new memory to matrix
  - Mat.**cross**(<Mat>) - computes cross product of two matricies
  - Mat.**depth**() - returns data type of matrix
  - Mat.**dot**(<Mat>) - computes the dot product of two matrices

cv::Mat

- Functions
  - Mat(Range(xmin,xmax),Range(ymin,ymax)) - returns sub image
  - Mat.**type**() - returns the TYPE of a matrix
- Iterator Usage
  - Mat.**begin**() - moves Mat iterator to beginning of image
  - Mat.**end**() - moves Mat iterator to end of image

```
34 //Example of using iterators to invert an image
35 MatConstIterator_<uchar> src_it = image.begin<uchar>();
36
37 MatConstIterator_<uchar> src_it end = image.end<uchar>();
38
39 MatIterator_<uchar> dst_it = ret.begin<uchar>();
40
41 for(; src it != src it end; src it++,dst it++){
42   pix = *src it;
43   *dst it = uchar(255) - pix;
44 }
```

Image TYPES

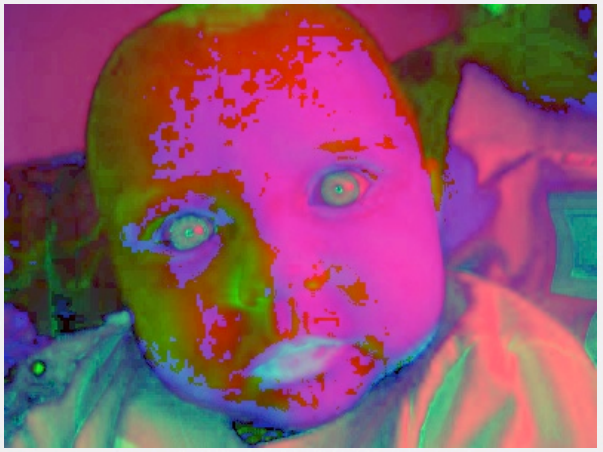
- The TYPE is a very important aspect of OpenCV
- Represented as CV_<Datatype>C<# Channels>
- Example Datatypes/ Depths

OpenCV Tag	Representation	OpenCV Value
CV_8U	8 bit unsigned integer	0
CV_8S	8 bit signed integer	1
CV_16U	16 bit unsigned integer	2
CV_16S	16 bit signed integer	3
CV_32S	32 bit signed integer	4
CV_32F	32 bit floating point number	5
CV_64F	64 bit floating point number	6

Pixeltypes

- PixelTypes shows how the image is represented in data
  - BGR - The default color of imread(). Normal 3 channel color
  - HSV - Hue is color, Saturation is amount, Value is lightness. 3 channels
  - GRAYSCALE - Gray values, Single channel

- OpenCV requires that images be in BGR or Grayscale in order to be shown or saved. Otherwise, undesirable effects may


HELLO WORLD

- Example Code

```
//Loads image and displays
//call by ./a.out image.jpg
//
#include <cv.h>
#include <cvaux.h>
#include <highgui.h>

using namespace cv;

int main(int argc, char* argv[ ]){
   Mat image = imread(argv[1]);

   namedWindow("Sample Window");
   imshow("Sample Window",image);
   waitKey(0);
   return 0;
}
```

This program will load and show an image

Starting Out in OpenCV

- OpenCV uses the **cv** namespace.
- cv::Mat object replaces the original C standard **IplImage** and **CvMat** classes.
- All original functions and classes of the C standard OpenCV components in the Bradski book are still available and current. However you will need to read that book for it.
- **namedWindow** is used for viewing images. See my manual for instructions on calling it.
  - In general, default string as input with original image size set. Else, use string as input name and 0 for adjustable size.

Image I/O

- OpenCV provides simple and useful ways to read and write images.
- Note that there are many extra options to these commands which are available on the wiki.
- waitKey(int x) has two main features.
  - if x > 0, then waitKey will wait x milliseconds
  - if x = 0, then waitKey will not move until key is pressed

- Examples

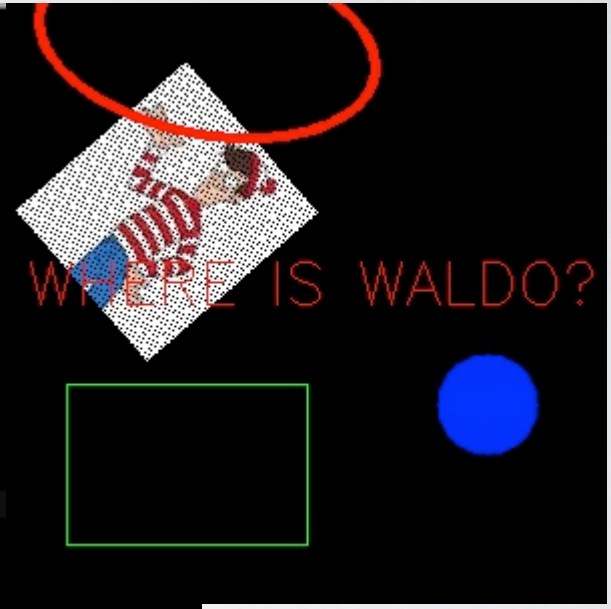
```
//Read an image
Mat image = imread( <string>, <0 -gray, 1 -BGR>)
      //Note 1 is default

//Write an image
imwrite( <string filename> , image );

//Create window for output
namedWindow( <window name> );

//Output image to window
imshow( <window name> , <image Mat to show> );

//pause program for input
key = waitKey( 0 );
```

DRAWING STUFF

- Sometimes it is necessary to draw stuff onto the image. Instead of using complicated functions, why not just call a simple function?
- Here are some simple examples...
- void **circle**(image, Point(x,y),int rad, CV_BGR(b,g,r), int thickness=1)
- void **ellipse**(image, RotatedRect box, CV_BGR(b,g,r), int thickness=1)
- void **line**(image, Point(x,y), Point(x,y), CV_BGR(b,g,r), int thickness= 1)
- void **rectangle**(img, Point(x,y), Point(x,y), CV_BGR(b,g,r), int thickness)
  - NOTE: negative thickness will fill in the rectangle
- MORE... http://opencv.willowgarage.com/documentation/cpp/core_drawing_functions.html

Drawing stuff

```cpp
#include <cv.h>
#include <cvaux.h>
#include <highgui.h>

using namespace cv;

int main(int argc, char* argv[]){

  Mat image(300,300,CV_8UC3);
  Mat sub = imread(argv[1]);
  float x,y;

  //Project image onto new with 45deg rotation
  for(int i=0;i<sub.rows;i++)
    for(int j=0;j<sub.cols;j++){
      x = (j+0)*cos(0.85398)-(i-0)*sin(0.85398);
      y = (j+0)*sin(0.85398)+(i-0)*cos(0.85398);
      if(x+90 >= 0 && y+30 >= 0 && x+90 < image.cols && y+30 < image.rows)
        image.at<Vec3b>(y+30,x+90) = sub.at<Vec3b>(i,j);
    }

  //Draw an ellipse
  RotatedRect rotrect(Point(100,20),Size(90,170),101);
  ellipse(image,rotrect,Scalar(0,0,255),3);

  //Draw a circle
  circle(image,Point(240,200),25,Scalar(255,0,0,0),-1);

  //Draw a box
  rectangle(image,Point(30,190),Point(150,270),Scalar(0,255,0),1);

  //Place Text
  putText(image,"WHERE IS WALDO?",Point(10,150),FONT_HERSHEY_SIMPLEX,1,Scalar(0,0,255));

  //Output
  imwrite("image0.jpg",image);

  return 0;
}
```

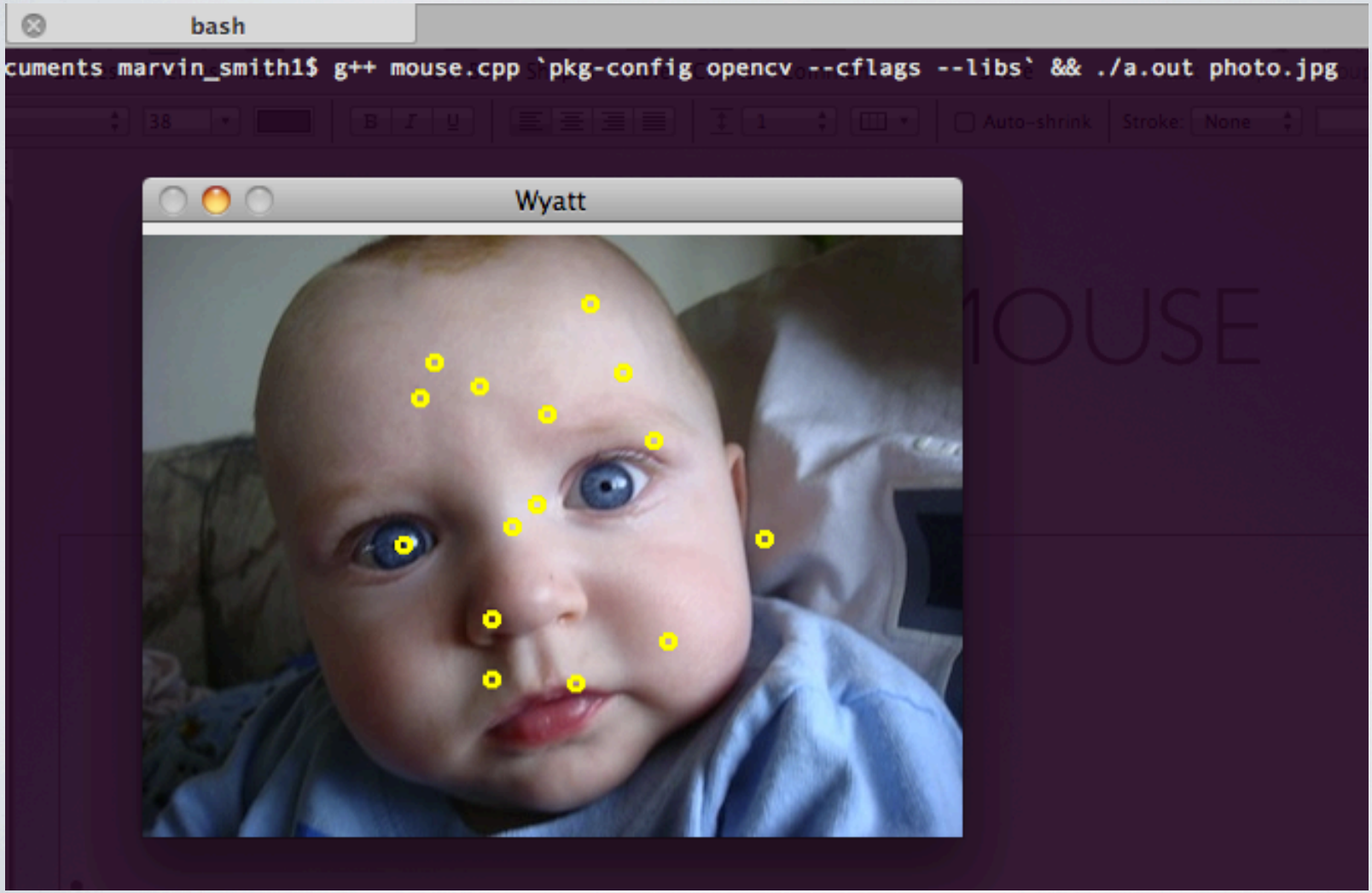


Using the Mouse

- OpenCV allows you to use the mouse to interact with the screen. Note that this feature is from OpenCV 1.0 and is compatible with Mat objects.
- This program allows you to draw dots on the image.

```
7  struct OPTIONS{
8    OPTIONS(): X(-1),Y(-1),drawing_dot(false){}
9    int X;
10   int Y;
11   bool drawing_dot;
12 };
13 OPTIONS options;
14
15 void my_mouse_callback( int event, int x, int y, int flags, void* param ){
16   IplImage* image = (IplImage*) param;
17
18   switch( event ){
19
20     case CV_EVENT_LBUTTONDOWN:
21       options.X = x;
22       options.Y = y;
23       options.drawing_dot = true;
24       break;
25   }
26 }
27
28 int main(int argc, char* argv[]){
29
30   IplImage* image = cvLoadImage(argv[1]);
31   Mat frame = imread(argv[1]);
32
33   namedWindow("Wyatt");
34   cvSetMouseCallback("Wyatt", my_mouse_callback, (void*) image);
35
36   //Take new points from user
37   while(cvWaitKey(15) != 27){
38     if( options.drawing_dot ){
39
40       circle(frame,Point(options.X,options.Y),3,CV_RGB(255,255,0),2);
41       options.drawing_dot = false;
42     }
43
44     imshow("Wyatt",frame);
45     waitKey(10);
46   }
47   cvReleaseImage(&image);
48
49   return 0;
```

USING THE MOUSE



Converting colorspaces

- **cvtColor**(image, image, code)
  - Codes
    - CV_<colorspace>2<colorspace>
    - Examples
    - **CV_BGR2GRAY**
    - **CV_BGR2HSV**
    - **CV_BGR2LUV**

Image Normalization

- **normalize**(imagein, imageout, low, high, method);
- Image normalization is the process of stretching the range of an image from [a, b] to [c, d].
- This is incredibly important for visualization because if the image is beyond [0,255] it will cause truncation or unsightly effects.

```
#include <cv.h>
#include <cvaux.h>
#include <highgui.h>

#include <iostream>

using namespace cv;
using namespace std;

int main(int argc, char* argv[]){

  Mat image = imread(argv[1],0);
  Mat data, dx, dy;
  float pix;

  imwrite("image_0.jpg",image);

  image.convertTo(data,CV_32FC1);
  data = data*4;

  Sobel(data,dx,CV_32FC1,1,0);
  Sobel(data,dy,CV_32FC1,0,1);

  MatConstIterator_<float>dx_it     = dx.begin<float>();
  MatConstIterator_<float>dx_it_end = dx.end<float>();
  MatConstIterator_<float>dy_it     = dy.begin<float>();
  MatIterator_<float> dst_it        = data.begin<float>();

  for(; dx_it != dx_it_end; dst_it++,dx_it++,dy_it++){
    *dst_it = sqrt(pow(*dx_it,2)+pow(*dy_it,2));
  }

  data.convertTo(image,CV_8UC1);
  imwrite("image_1.jpg",image);

  normalize(data,data,0,255,CV_MINMAX);
  data.convertTo(image,CV_8UC1);
  imwrite("image_2.jpg",image);

  return 0;
}
```

Thresholding

- threshold(image, image, thresh, maxVal, CODE);
- CODE - this is the method of thresholding. Different actions will be taken depending on this code.

- **THRESH_BINARY**

$$\texttt{dst}(x,y) = \begin{cases} \texttt{maxVal} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

- **THRESH_BINARY_INV**

$$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{maxVal} & \text{otherwise} \end{cases}$$

- **THRESH_TRUNC**

$$\texttt{dst}(x,y) = \begin{cases} \texttt{threshold} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$$

- **THRESH_TOZERO**

$$\texttt{dst}(x,y) = \begin{cases} \texttt{src}(x,y) & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

- **THRESH_TOZERO_INV**

$$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$$

Edge Detection

- Sobel Edge Detection

void cv::Sobel(image in, image out, CV_DEPTH, dx, dy);

- Scharr Edge Detection

void cv::Scharr(image in, image out, CV_DEPTH, dx, dy);

- Laplacian Edge Detection

void cv::Laplacian(image in, image out, CV_DEPTH);

```
3  #include <cv.h>
4  #include <cvaux.h>
5  #include <highgui.h>
6
7  using namespace cv;
8
9  int main(){
10
11   Mat img = imread("photo.jpg",0);
12
13   imwrite("photo_gray.jpg",img);
14   Mat img_float;
15   img.convertTo(img_float,CV_32FC1);
16
17   Sobel(img_float,img_float,CV_32FC1,0,1);
18
19   Mat img_abs = abs(img_float);
20
21   normalize(img_float,img_float,0,255,CV_MINMAX);
22   normalize(img_abs  ,img_abs  ,0,255,CV_MINMAX);
23
24   img_float.convertTo(img,CV_8UC3);
25   imwrite("photo_float.jpg",img);
26   img_abs.convertTo(img,CV_8UC3);
27   imwrite("photo_abs.jpg",img);
28
29   return 0;
30 }
```

Image Smoothing

- Image smoothing is used to reduce the the sharpness of edges and detail in an image.
- OpenCV includes most of the commonly used methods.

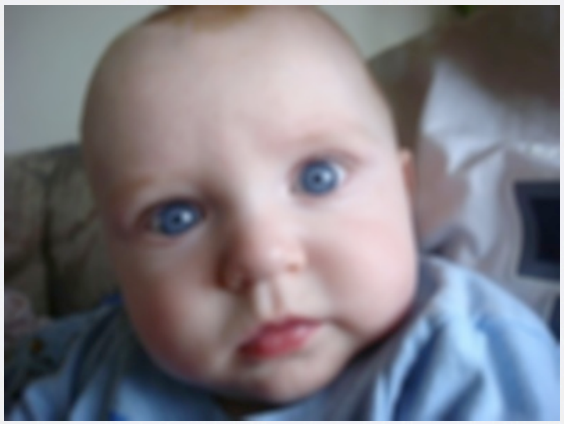
- void **`GaussianBlur`**(imagein, imageout, Size ksize, *sig);*
  - *Note that there are more options, however this should keep things simple*

- *void* **`medianBlur`** (imagein, imageout, Size ksize);

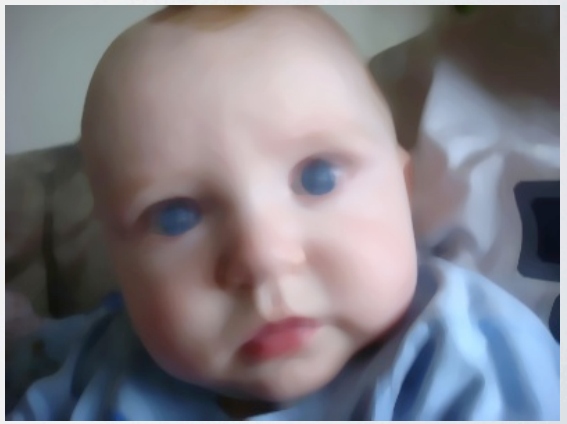
- Other functions include generic convolution, separable convolution, dilate, and erode.

```
#include <cv.h>
#include <cvaux.h>
#include <highgui.h>

using namespace cv;

int main(int argc, char* argv[]){

   Mat image = imread(argv[1],1);
   Mat data1, data2;
   imwrite("image0.jpg",image);

   GaussianBlur(image,data1,Size(7,7),3);
   medianBlur(image,data2,11);

   imwrite("image1.jpg",data1);
   imwrite("image2.jpg",data2);

   return 0;
}
```

Original



Gaussian Blur



Median Blur

STOP!

This is not relevent until the last part of the class.

Beware!

Linear Algebra

- OpenCV contains many useful and simple functions for applying linear algebra on images.
- Most major operators are allowed.
- operator * performs matrix multiplication, NOT elementwise multiplication.

Operators

given: Mat image;

- **image.inv();** //inverse
- **image.t();** //transpose
- **image.clone();** //creates deep copy
- **image.diag(int d=0)** //returns diagonal
- **image.mul(mat, double);** //performs elementwise multiplication.
- **image.cross(mat);** //performs cross product
- **image.dot(mat);** //performs dot product
- **image.eye();** //converts mat to identity matrix

Singular Value Decomposition

- Singular Value Decomposition is a vital part of any computer vision based system. Luckily, OpenCV makes this a trivial task.
- To solve a least-squares problem, simply call the **solve** command.
- **bool solve(src1, src2, dst, int flags);**
- Usually, src1 is A, src2 is b, and dst is x. Remember flags is method...
- **DECOMP_LU** - Fast but cannot solve over-determined systems.
- **DECOMP_SVD** - SVD, can solve just about anything
- Others available, but stick to the basics...

Example

given:

$$-11x + 2y = 0$$
$$2x + 3y = 7$$
$$2x - y = 5$$

```
//Main Driver
int main(){

  Mat data1(3,3,CV_32FC1);
  Mat data2(3,1,CV_32FC1);
  Mat results;

  //Matrix A
  data1.at<float>(0,0) = -11;
  data1.at<float>(0,1) =   2;
  data1.at<float>(1,0) =   2;
  data1.at<float>(1,1) =   3;
  data1.at<float>(2,0) =   2;
  data1.at<float>(2,1) =  -1;

  //Matrix b
  data2.at<float>(0,0) = 0;
  data2.at<float>(1,0) = 7;
  data2.at<float>(2,0) = 5;

  solve(data1,data2,results,DECOMP_SVD);

  Print_Mat(results);

  return 0;
}
```

SVD Results

- Using OpenCV

```
Marvin-Smiths-MacBook-Pro:Documents marvin_smith1$ g++ SVD.cpp `pkg-config opencv --libs --cflags`
Marvin-Smiths-MacBook-Pro:Documents marvin_smith1$ ./a.out
x = 0.421053
y = 1.68421
Marvin-Smiths-MacBook-Pro:Documents marvin_smith1$
```

- Using GNU Octave

```
octave:3> A = [-11 2; 2 3; 2 -1];
octave:4> b = [0; 7; 5];
octave:5>
octave:5> A\b
ans =

   0.42105
   1.68421

octave:6>
```

Principle Component Analysis

- Since you will need to learn this, I will include it. Although you will undoubtably will have to create your own PCA program, OpenCV covers it very nicely.

- PCA(Mat data, Mat mean, int FLAG, int numcomp=0)
  - FLAG: PCA_DATA_AS_ROW / PCA_DATA_AS_COL
  - numcomp is the k value, 0 means all values retained
  - in general, just pass the vectors into data and the mean will be returned.

- PCA.project(Mat vector)
  - projects the vector into the built eigenspace and returns the result

- PCA.backproject(Mat vector)
  - reconstructs the vector from the principle component subspace

Important Tips

- Remember that images are read from file as 8-bit unsigned integers. In order to do complicated math operations, convert to 32-bit floating point type. Then convert back to write to file.

- Always remember that rows is your y coordinate and that cols is your x coordinate. Size objects are called X,Y while images are referenced row, col. There are many subtle things that will ruin good code.