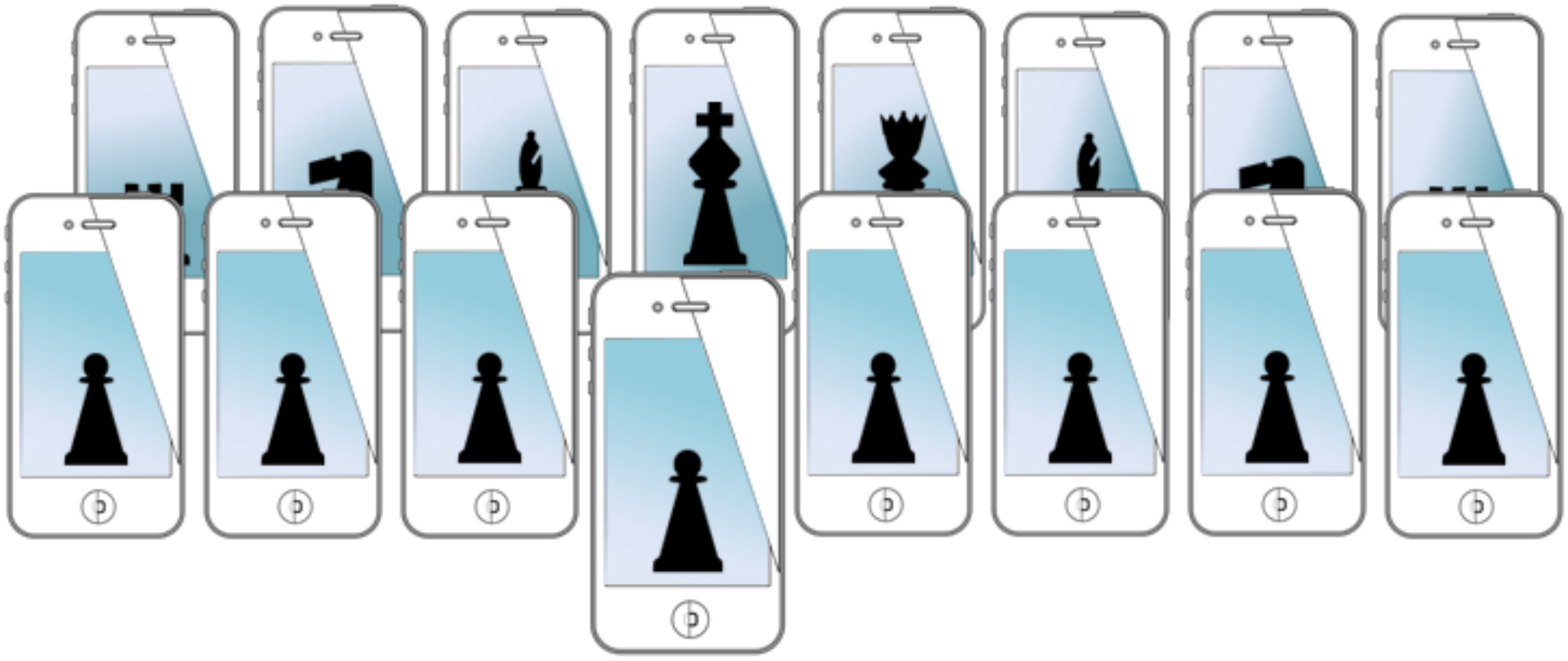
MOBILE SENSING LEARNING



# CSE5323 & 7323

## Mobile Sensing and Learning

week 7: computer vision with OpenCV

Eric C. Larson, Lyle School of Engineering,
Computer Science and Engineering, Southern Methodist University

# video agenda

- OpenCV in iOS
  - we will look at **using** the tool
  - focus on
    - **outputs** of each algorithm
    - **how to use** each method
- **ignore** most of what is **under the hood**

# OpenCV in iOS

- open computer vision library
- released by intel
- many common functions are implemented
- **written in c++**, but has many wrappers
  - EMGU for .NET (c#,VC++,etc.), pycv2 for python, Java API for Android, and many, many more (**but not swift**)
- some hardware accelerations on iOS
  - but not the GPU, **only accelerate** functions
  - expect **slower** processing, but still **pretty fast**!

# OpenCV installation

- download the opencv framework for iOS
  - might need to build from scratch… (or not)
- drag into project
- manually add a bunch of dependencies
- step by step instructions:
  - http://docs.opencv.org/doc/tutorials/ios/video_processing/video_processing.html
- remember to rename your model to .mm
- **alternatively**: do a git checkout of **ImageLab**

# OpenCV video

- online tutorials will show you how to setup video capture
  - you can use the delegate protocol and a cvVideoCamera
- or keep using **VideoAnalgesic**!

```
let bridge = OpenCVBridge()

self.bridge.setImage(retImage, withBounds: retImage.extent, andContext:
                     self.videoManager.getCIContext())
self.bridge.processImage()
retImage = self.bridge.getImage()

let f = getFaces(inputImage)
self.bridge.setImage(retImage,
                     withBounds: f[0].bounds, // the first face
                     andContext: self.videoManager.getCIContext())

self.bridge.processImage()
retImage = self.bridge.getImageComposite() // get back (overlayed on original)
```

not deprecated anymore!

# OpenCV video

- now head over to OpenCVBridge.mm

```
-(void)processImage{

    cvtColor( _image, frame_gray, CV_BGR2GRAY );
    bitwise_not(frame_gray, _image);


}
```

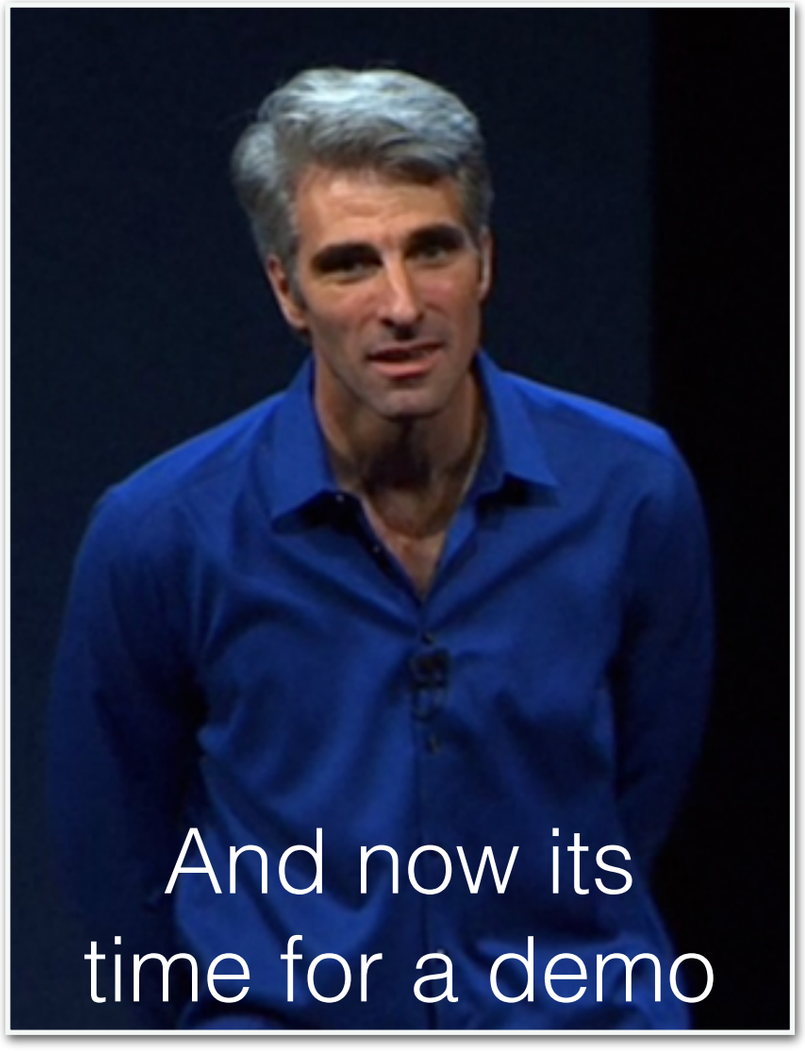
```
-(void)processImage{

    cvtColor(_image, image_copy, COLOR_BGRA2GRAY);
    cv::threshold(image_copy, image_copy, 0, 255, CV_THRESH_BINARY | CV_THRESH_OTSU);
    cvtColor(image_copy, _image, CV_GRAY2BGRA); //add back for display


}
```

# OpenCV video

- you are not on the main queue here
- I am using an older version of OpenCV 2
  - but you can build OpenCV 3.0 and use it if you want!
- OpenCV is **mostly updated** for iOS
  - some functions are deprecated, but currently work, some transforms are not quite right

```
-(void)processImage{

    cvtColor( _image, frame_gray, CV_BGR2GRAY );
    bitwise_not(frame_gray, _image);


}
```

# OpenCV Setup Demo




# access torch

before we get too far, some convenience methods

```
@IBAction func toggleFlash(sender: AnyObject) {
     self.videoManager.toggleFlash()
 }
```

```
 @IBAction func setFlashLevel(sender: UISlider) {
        if(sender.value>0.0){
            self.videoManager.turnOnFlashwithLevel(sender.value)
        }
        else if(sender.value==0.0){
            self.videoManager.turnOffFlash()
        }
    }
```

# OpenCV operations

- your input is a matrix
- data is interleaved BGR
  - if setting color conversion to CV_BGRA2BGR
  - must get a pointer in the array for the row and column

```
image.ptr(row, column)
```

starts from upper left

```
for(int i=0;i<50;i++){
      image.ptr(i, i)[0] = 255;
      image.ptr(i, i)[1] = 0;
      image.ptr(i, i)[2] = 0;
 }
```

blue

green

red

```
for(int i=0;i<50;i++){
      uchar *pt = image.ptr(i, i);
      pt[0] = 255;
      pt[1] = 0;
      pt[2] = 0;
 }
```

```
for(int i=0;i<50;i++){
      uchar *pt = image.ptr(i, i);
      pt[0] = 255;
      pt[1] = 0;
      pt[2] = 0;
      pt[3] = 255;
      pt[4] = 0;
      pt[5] = 0;
 }
```

pixel at i,i

next pixel in row

# OpenCV operations

- filtering

```
Mat gauss = cv::getGaussianKernel(25, 3);
cv::filter2D(image_copy, image_copy, -1, gauss);
GaussianBlur(image_copy, image_copy, cv::Size(3, 3), 2, 2 );
```

- inversion

```
bitwise_not(image_copy, image_copy);
```

- statistics

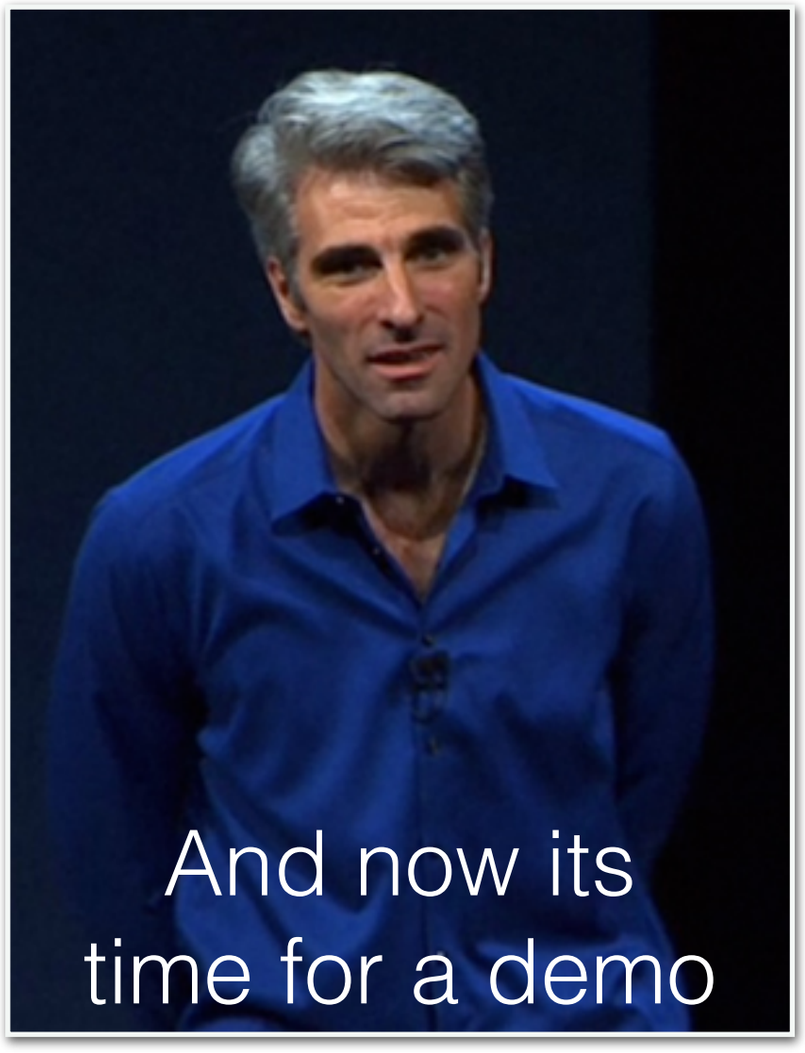
```
Scalar avgPixelIntensity = cv::mean( image_copy );
avgPixelIntensity.val[0]
avgPixelIntensity.val[1]
avgPixelIntensity.val[2]
```

- color conversion

```
cvtColor(image, image_copy, CV_BGRA2BGR);
cvtColor(image, image_copy, CV_GRAY2BGR);
cvtColor(image, image_copy, CV_RGB2BGR);
cvtColor(image, image_copy, CV_BGR2HSV);
cvtColor(image, image_copy, CV_BGR2Lab);
cvtColor(image, image_copy, CV_BGR2Lab);
cvtColor(image, image_copy, CV_BGR2YCrCb);
cvtColor(image, image_copy, CV_BGR2YUV);
```

# OpenCV Demo

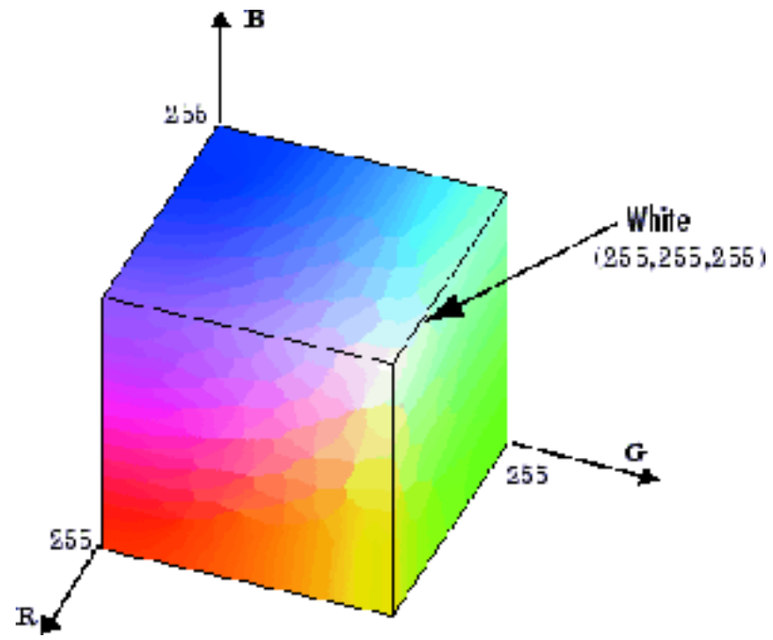
- basic operations

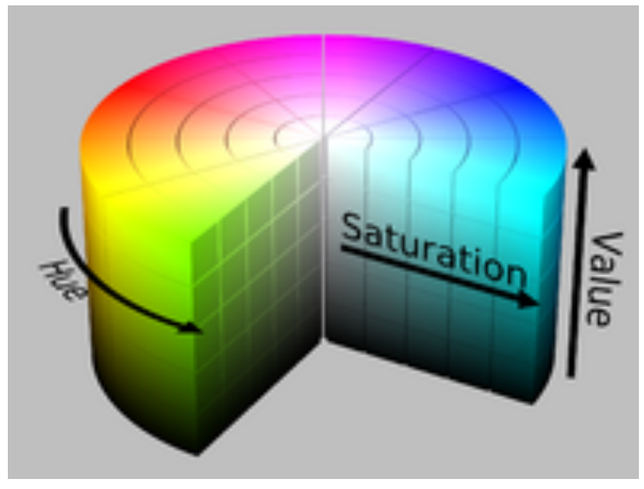



# color conversion

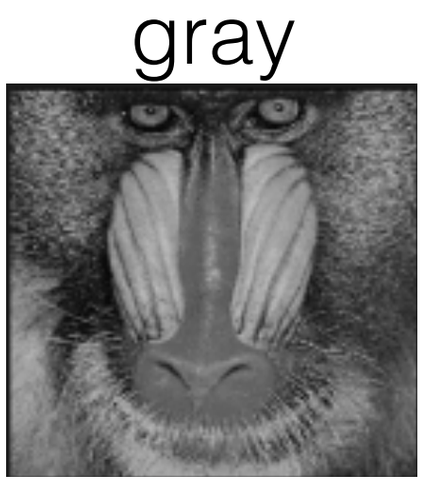
- to display properly, use BGRA

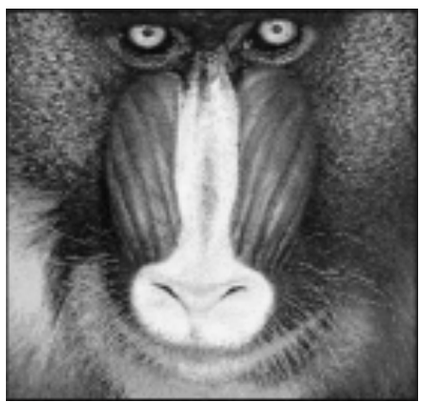
```
cvtColor(image, image_copy, CV_BGRA2BGR);
cvtColor(image_copy, image, CV_BGR2BGRA);

cvtColor(image, image_copy, CV_BGRA2GRAY);
cvtColor(image_copy, image, CV_GRAY2BGRA);

cvtColor(image, image_copy, CV_BGRA2HSV);
cvtColor(image_copy, image, CV_HSV2BGRA);
```

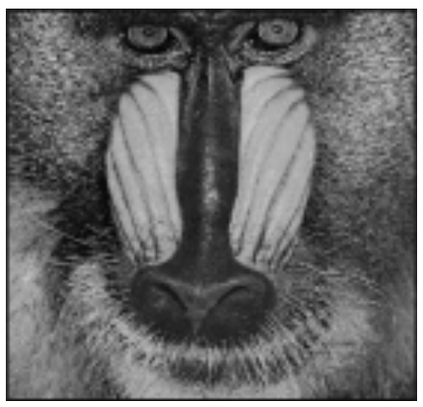


# color conversion

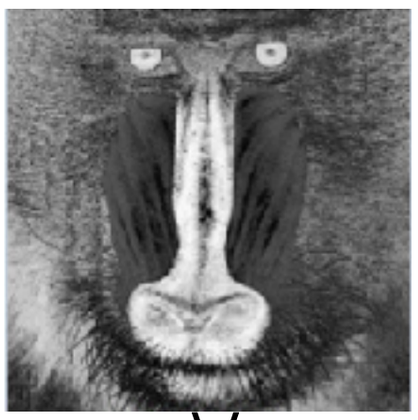



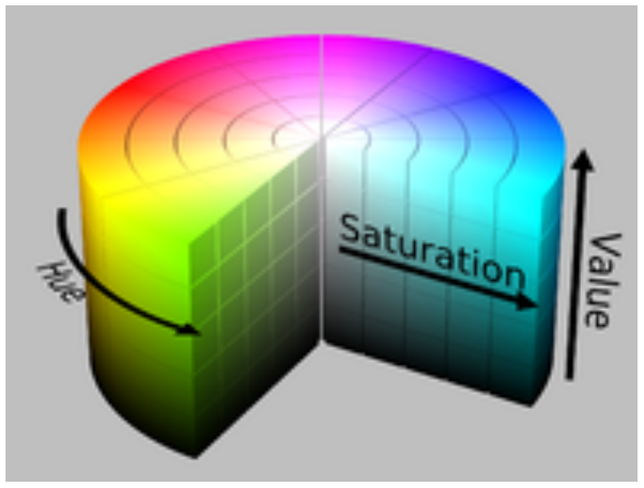










- **hsv**
  - what we perceive as color, rather than "sense" as color (sort of)
    - hue: the color value
    - saturation: the richness of the color relative to brightness
  - value: the intensity

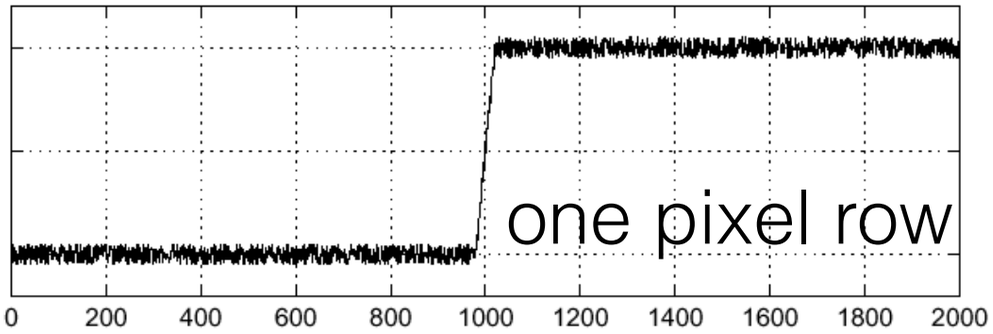


# edge detection

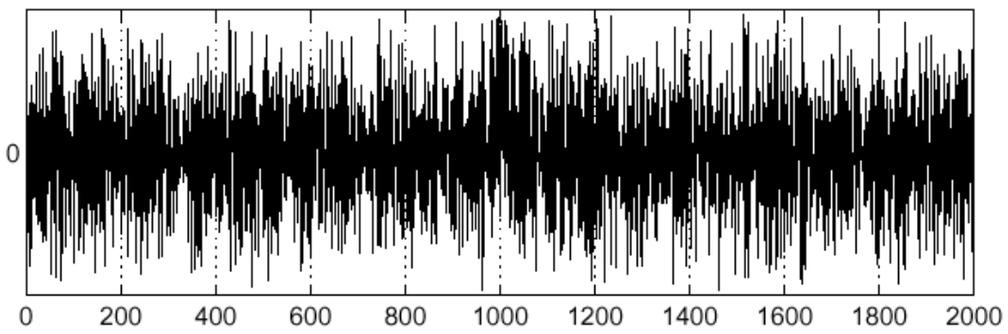
- can use linear filters to get gradient

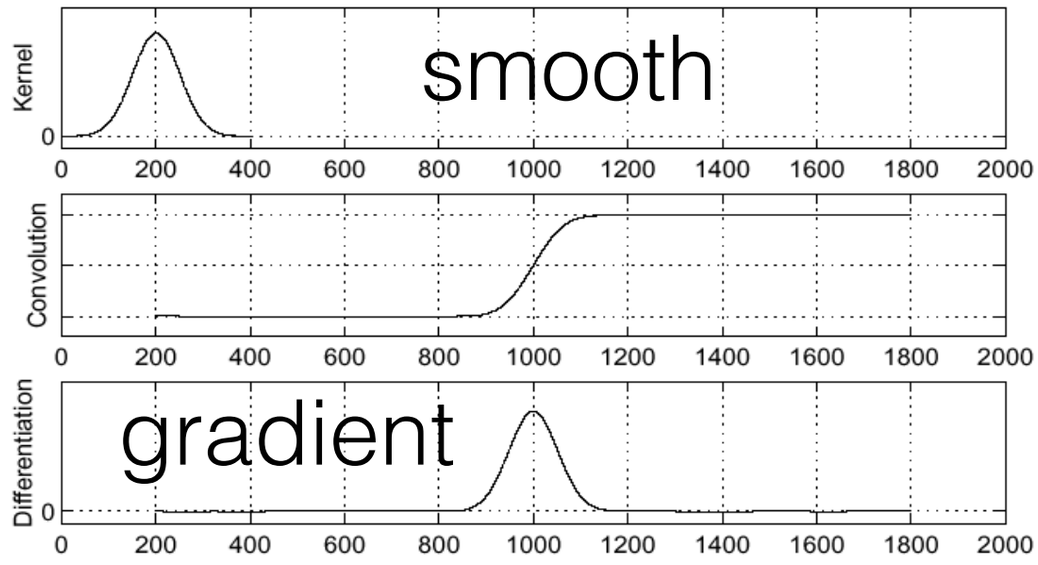
| | | |
|---|---|---|
| -1 | 0 | 1 |
| -2 | 0 | 2 |
| -1 | 0 | 1 |

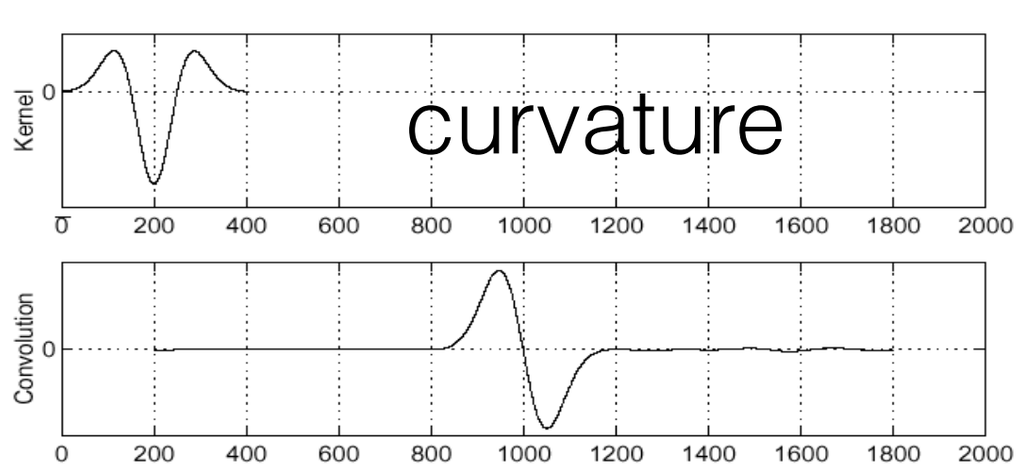
| | | |
|---|---|---|
| 1 | 2 | 1 |
| 0 | 0 | 0 |
| -1 | -2 | -1 |




$\frac{d}{dx} f(x)$




images courtesy of S. Narasimhan

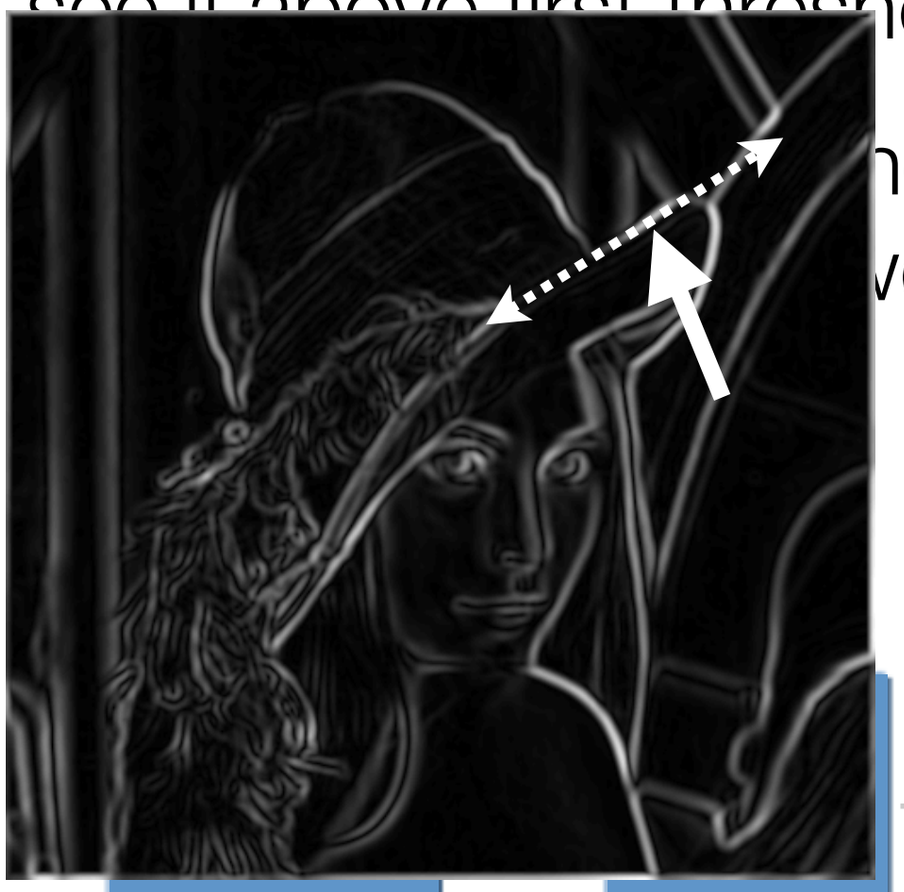
# gradient example

can we do better?

check local maxima

# canny edge detection

- get local maxima of gradient
- see if above first threshold
- [illegible] lar to [illegible]



$G_{x,y}$

From magnitude and direction of $G_{x,y}$

[illegible] sis [illegible] ding

hyst1 = image
hyst2 = along lines

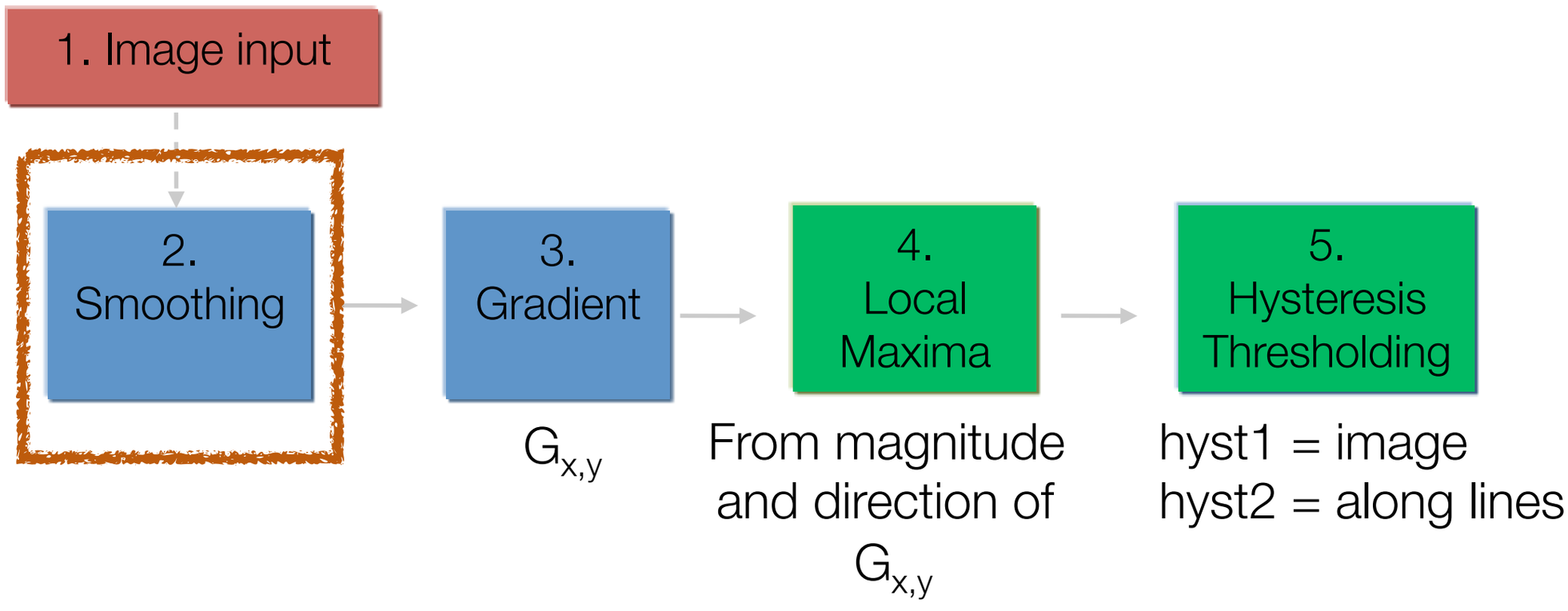
# canny edge detection

```
const int kCannyLowThreshold = 300;
cvtColor(image, grayFrame, COLOR_RGB2GRAY);

// Perform Canny edge detection using sl
Canny(grayFrame, output,
          kCannyLowThreshold,
          kCannyLowThreshold*7,
          kFilterKernelSize);

// copy back for further processing
cvtColor(output, image, CV_GRAY2BGRA); //add back for display
```

hyst2: small threshold

hyst1: large threshold

size of gradient filters



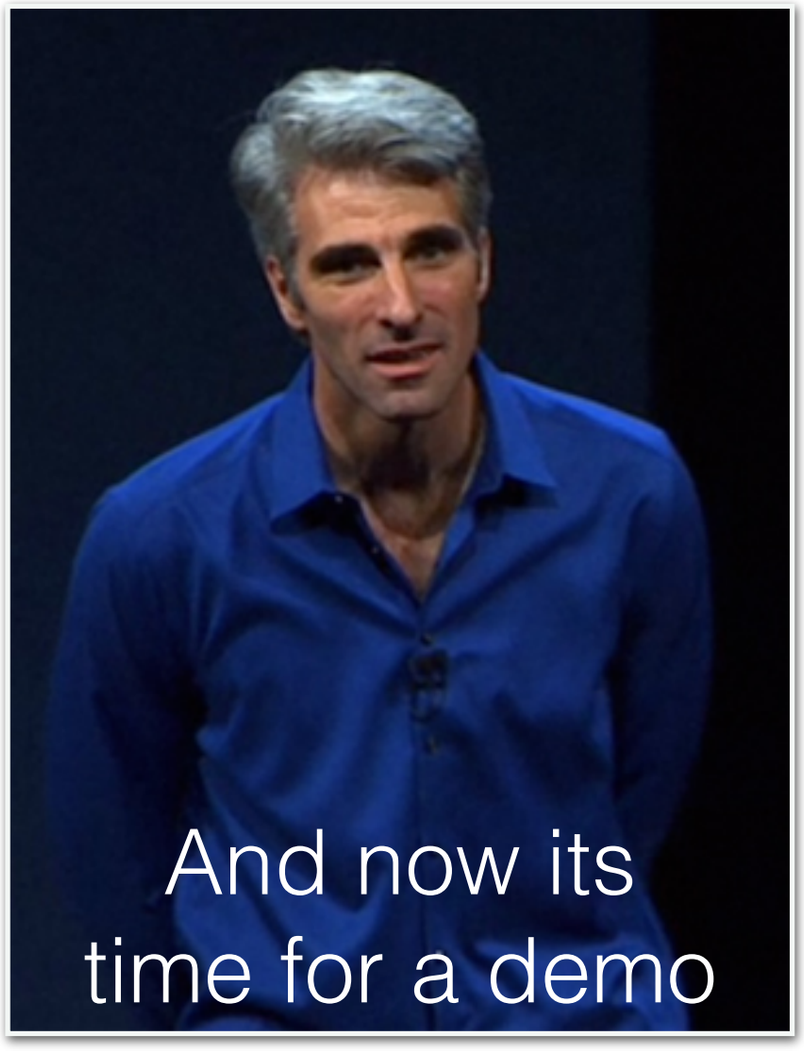
# edges to contours

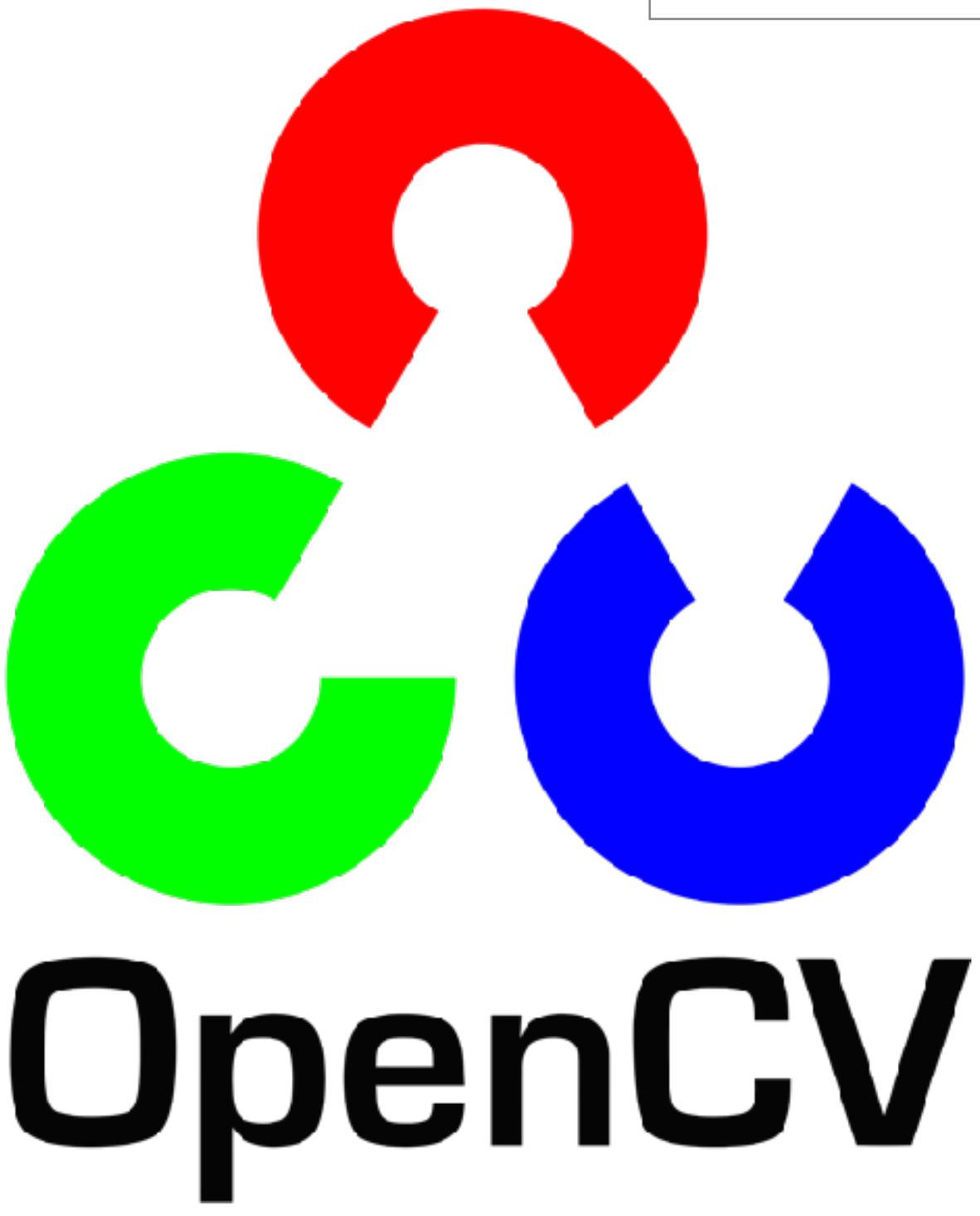
- connected components search
- contour detection from “outside” of component

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# OpenCV edge demo

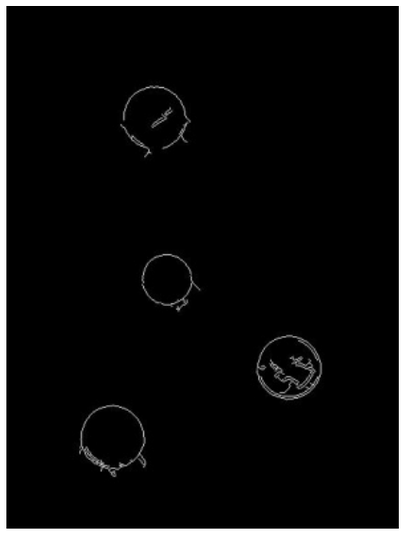
- edges
- contours

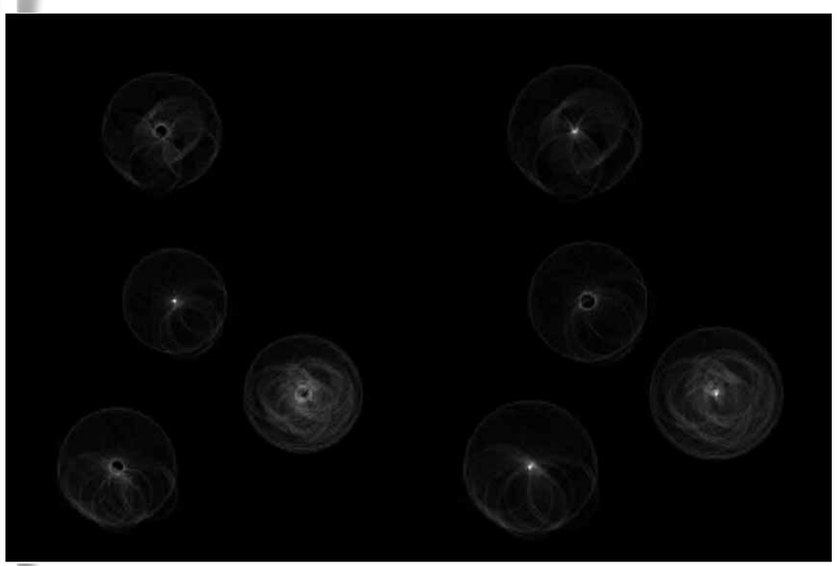


# hough transform

- In general, hough transform consists of
  - edge detection
  - for each detected point,
  - draw shape with different parameter
  - accumulation in parameter space
  - look for local maxima
  - for a circle, look for maxima in (x,y,R)





```
vector<Vec3f> circles;

/// Apply the Hough Transform to find the circles
HoughCircles( image_copy, circles,
              CV_HOUGH_GRADIENT,
              1, // downsample factor
              image_copy.rows/30, // distance between centers
              kCannyLowThreshold/2, // canny upper thresh
              40, // magnitude thresh for hough param space
              0, 0 ); // min/max centers
```

# generic Haar cascade

- remember Haar cascade filtering?
- you can use any trained cascade of classifiers
  - just need a trained file to load
- get some trained xml files, here is a start:
  - http://alereimondo.no-ip.org/OpenCV/34/
- or train your own (this is not trivial)
  - http://docs.opencv.org/doc/user_guide/ug_traincascade.html
  - database of positive example images
  - database of negative example images

# Haar syntax for iOS

```
cv::CascadeClassifier classifier;

// load in custom trained Haar Cascade filter
// This one is a famous trained face detector from Rainer Lienhart
 // http://www.lienhart.de/Prof._Dr._Rainer_Lienhart/Welcome.html
 NSString *fileName = [[NSBundle mainBundle]
                pathForResource:@"haarcascade_frontalface_alt2" ofType:@"xml"];

 classifier = cv::CascadeClassifier([fileName UTF8String]);
```

```
cvtColor(image, grayFrame, CV_BGRA2GRAY);
  vector<cv::Rect> objects;

  // run classifier
  classifier.detectMultiScale(grayFrame, objects);

  // display bounding rectangles around the detected objects
  for( vector<cv::Rect>::const_iterator r = objects.begin(); r != objects.end(); r++)
  {
      cv::rectangle( image,
            cvPoint( r->x, r->y ),
            cvPoint( r->x + r->width, r->y + r->height ),
            Scalar(0,0,255,255));
  }
```

# have fun with it!

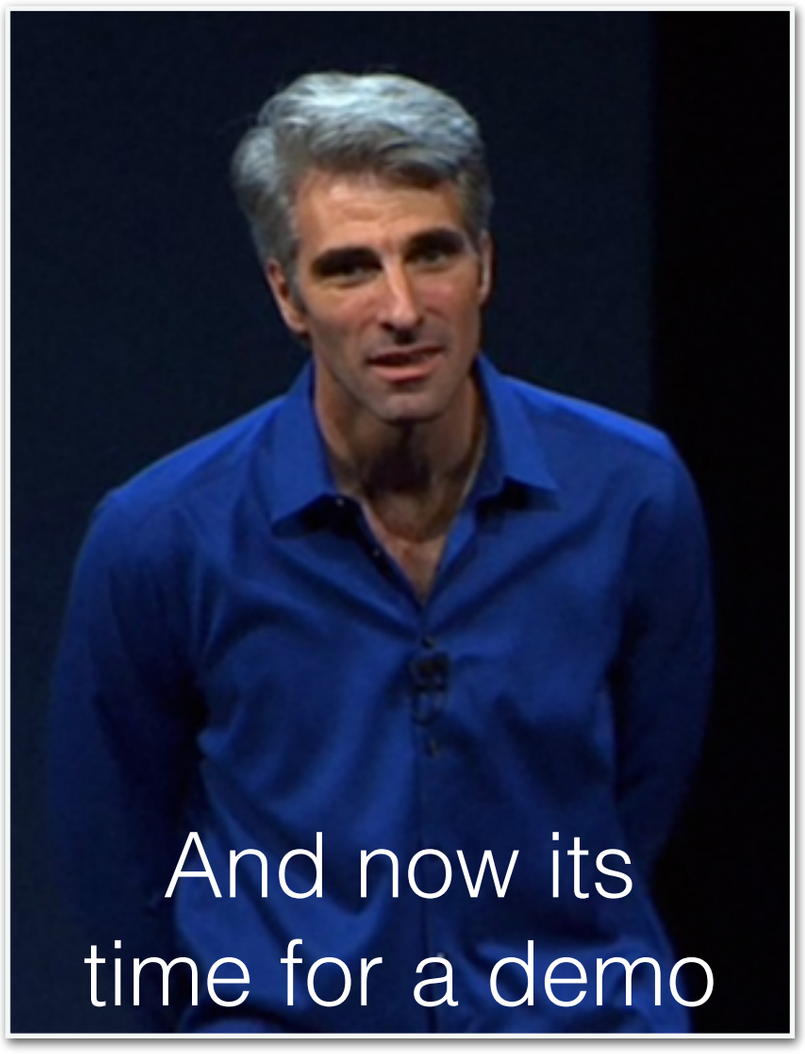
- OpenCV is a powerful framework
- many contributors
  - each algorithm has (possibly) different semantics
  - highly comprehensive and updated
  - lots of examples
    - OpenCV is absolutely an industry standard

# for class

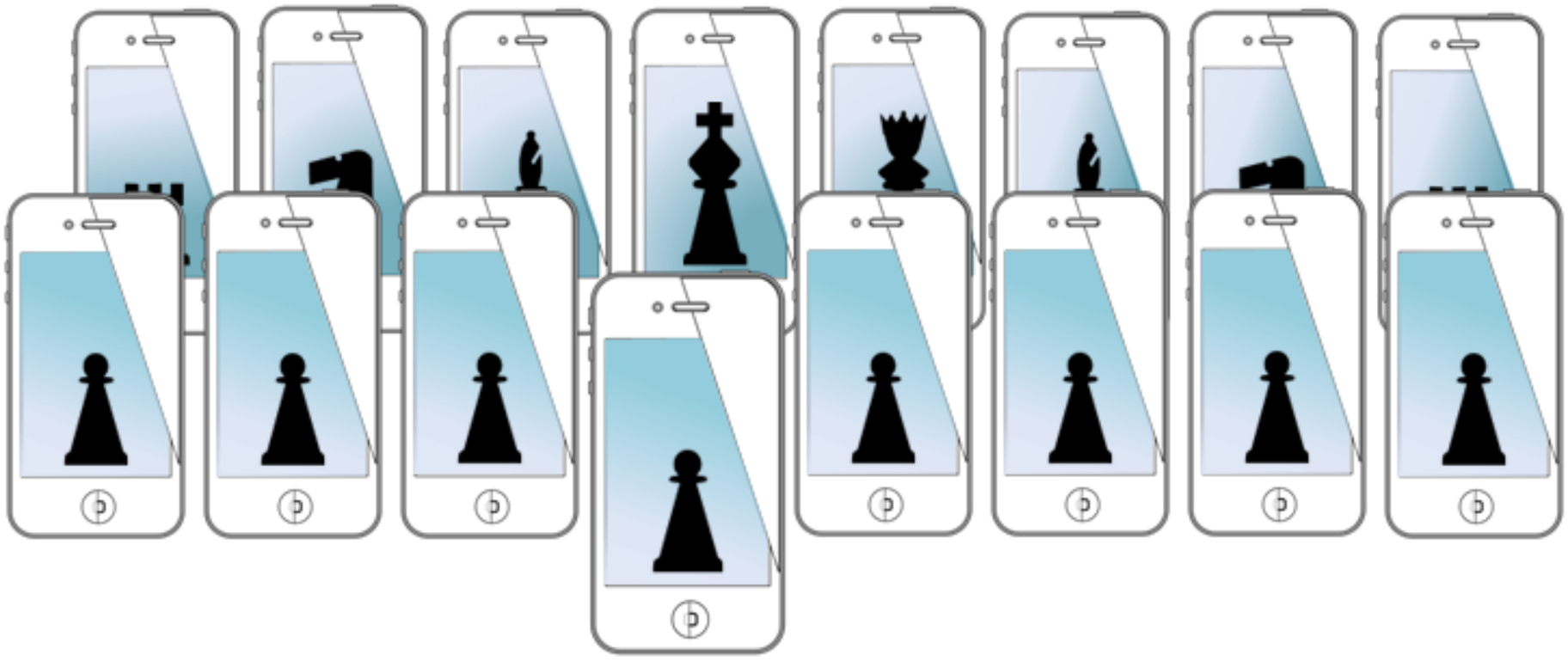
- download this example project from GitHub
- come ready to perform some computer vision

# OpenCV Demo

- pre-trained Haar cascades
- nose.xml




MOBILE SENSING LEARNING



# CSE5323 & 7323

## Mobile Sensing and Learning

week 7: computer vision with OpenCV

Eric C. Larson, Lyle School of Engineering,
Computer Science and Engineering, Southern Methodist University