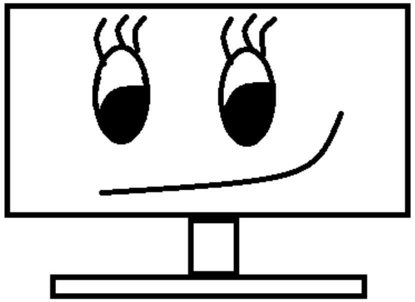
Seneca



CVI620/ DPS920 Introduction to Computer Vision

Digital Images

Seneca College

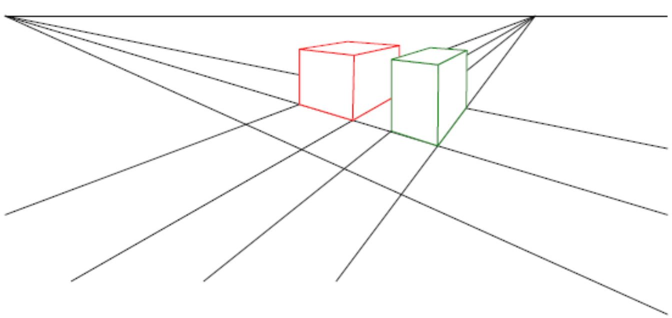
Vida Movahedi

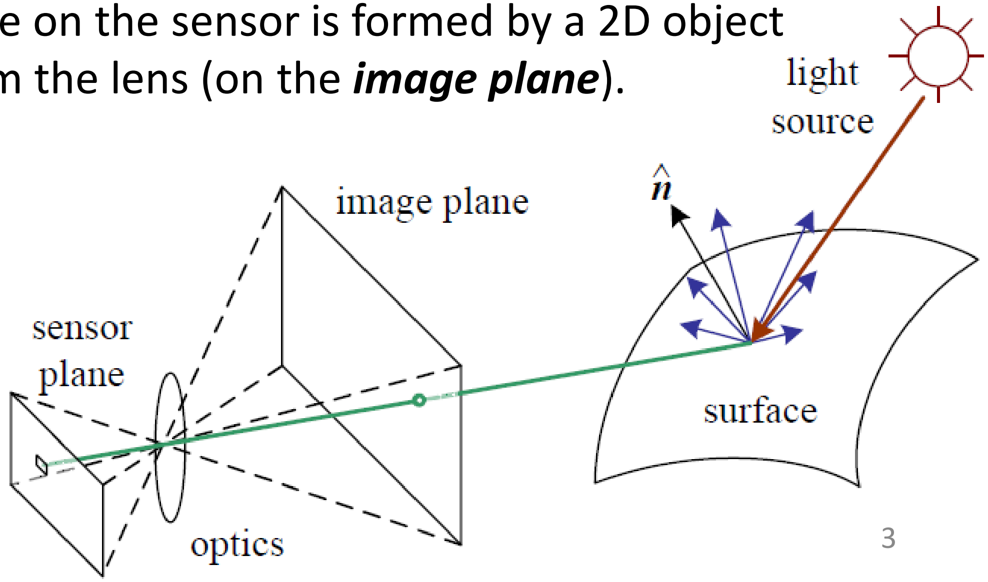
Overview

- Image Formation
- Digital Camera
- Digital Images and Image Representation
- Color & Compression
- OpenCV
  - Image files
  - Data types

Image formation [1]

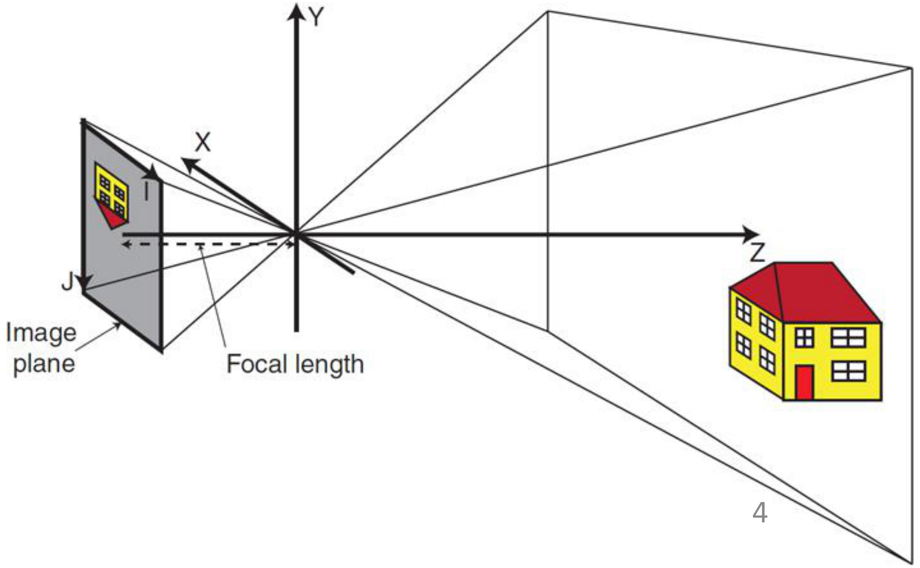
- Simplified model of photometric image formation
  - One (or more) light sources emit light
  - Light is reflected from an object's surface in different directions
  - A small portion of reflected light enters the camera
  - An image is formed on the sensor
- 3D properties of the object are transformed into 2D image features by *Perspective Projection.*
  - We can also assume the image on the sensor is formed by a 2D object existing at a unit distance from the lens (on the ***image plane***).





Cameras [3]

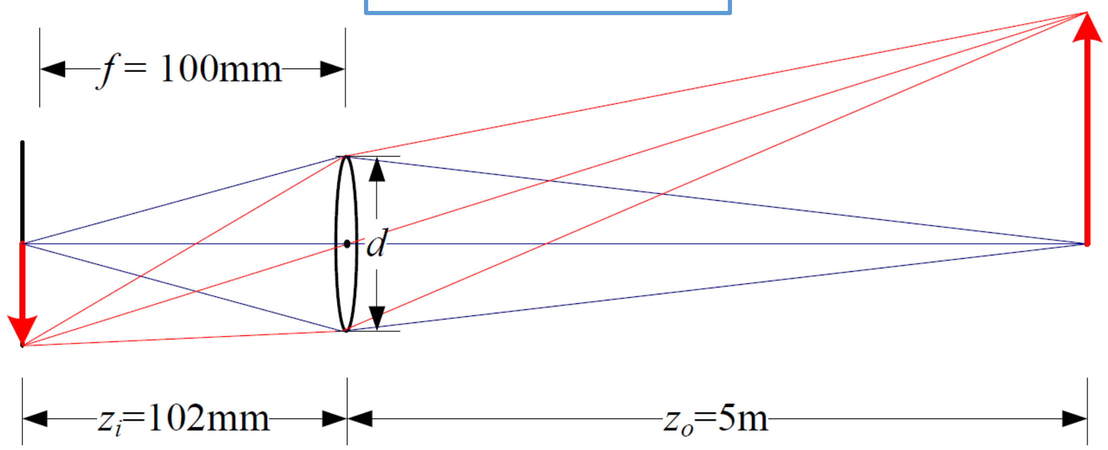
- A photosensitive image plane
- A housing (to prevent unwanted light)
- A lens, to focus light on the image plane
- Simple pinhole camera model
  - Assuming the lens as a simple pinhole



Optics

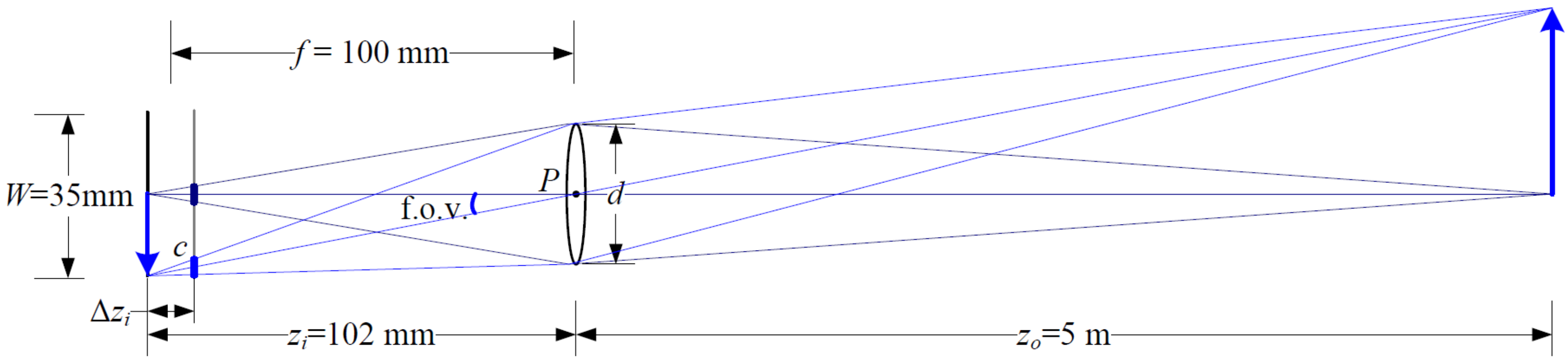
- Assuming the lens is an ideal pinhole
  - Focal length: *f*
  - Aperture diameter: *d*
- The light from a plane at distance z_o in front of the lens, is focused onto a plane at distance z_i behind the lens

$$\boxed{\frac{1}{z_o} + \frac{1}{z_i} = \frac{1}{f}}$$

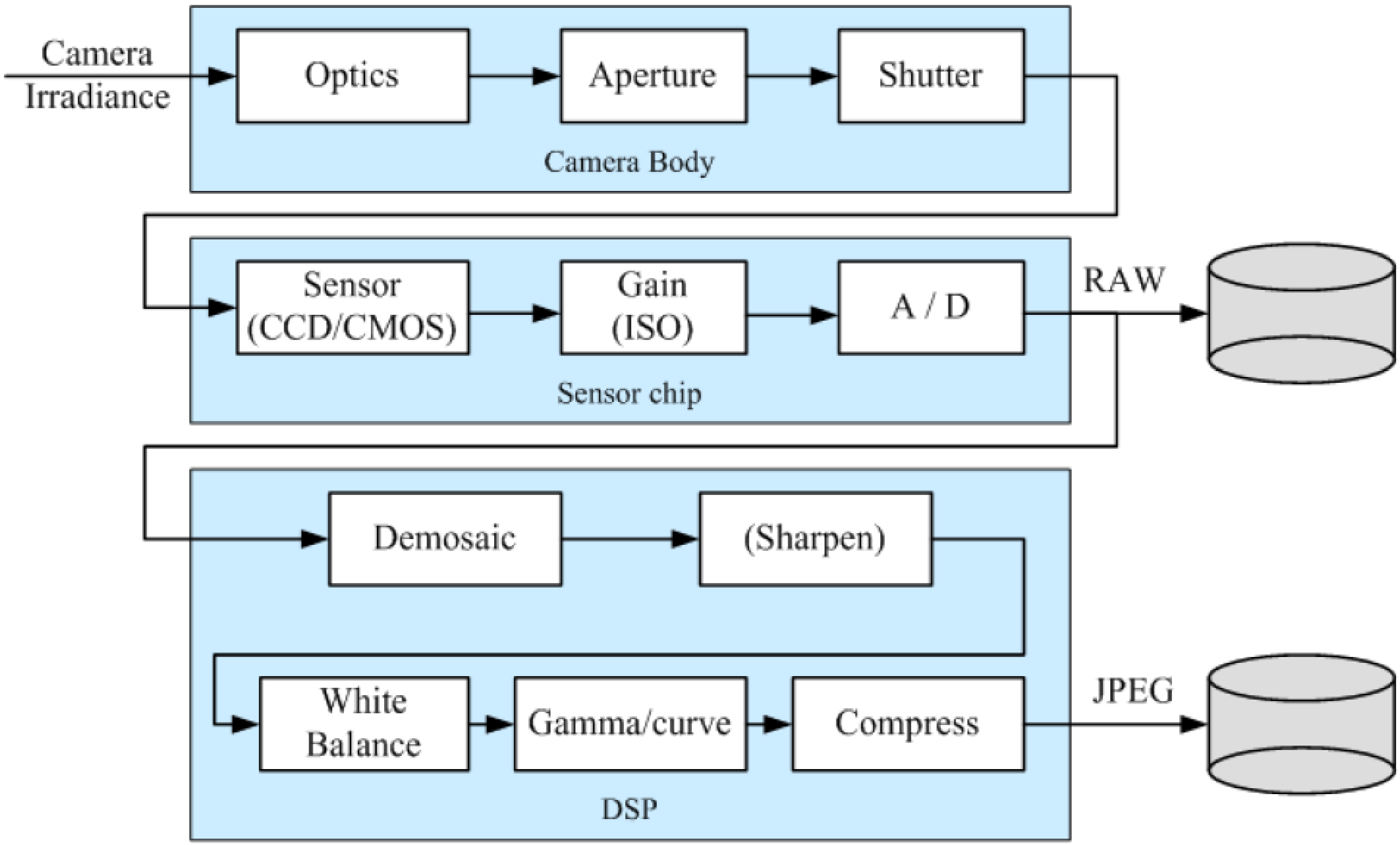


Out of focus

- If focal plane is not in its in-focus location, each point is imaged as a circle
- This is called ***circle of confusion*** (shown as *c*)



Digital Camera

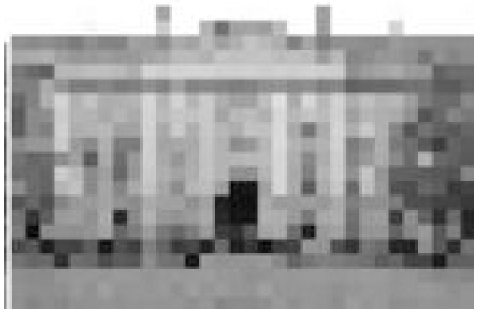


Digital Camera

- Shutter speed
  - What is the effect of a slow shutter speed?
- Sensors
  - CCD (charge-coupled device)
    - Suitable for quality sensitive applications
  - CMOS (complementary metal oxide on silicon)
    - Low power
    - Most digital cameras
- Camera Image formats
  - RAW: before digital processing or compression
  - Compressed (often JPEG): processed and compressed (often losing real color information)

Digital images [3]

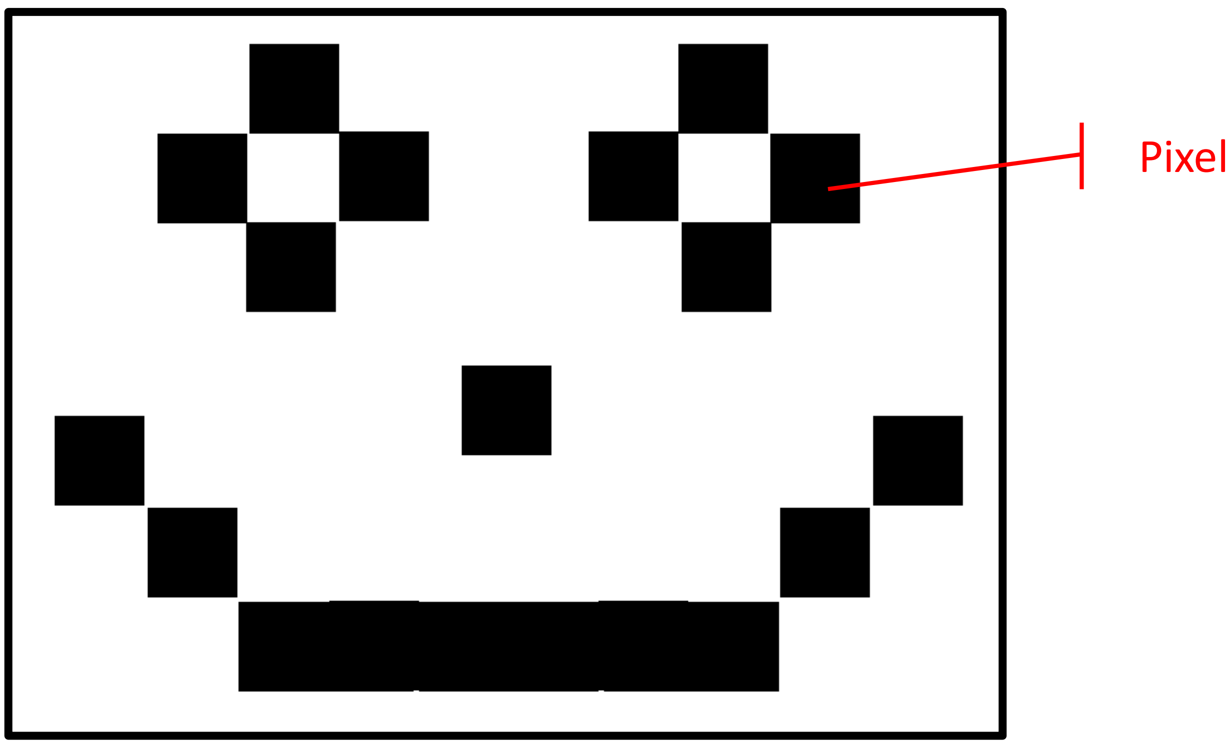
- Sampling
  - Samples of a continuous image into discrete elements
  - Resolution (number of elements)



- Quantization
  - Samples of continuous brightness values into discrete digital values
  - If b is the number of bits (often 8), the number of possible brightness levels is 2^b

Digital images: Bitmaps and pixels

- A **bitmap** is a 2-dimensional array of bits (0's and 1's)
- Also called a **binary** image
- Each element (of this array) is called a **pixel** (picture element)



Digital images: Grayscale image

- A **grayscale** image is an array of brightness intensity values
- Often expressed in 8 bits (0 to 255)

j $\rightarrow$

i $\downarrow$

62	79	23	119	120	105	4	0
10	10	9	62	12	78	34	0
10	58	197	46	46	0	0	48
176	135	5	188	191	68	0	49
2	1	1	29	26	37	0	77
0	89	144	147	187	102	62	208
255	252	0	166	123	62	0	31
166	63	127	17	1	0	99	30

Pixel

Digital images: Color and Beyond

- A **color** image consists of three color maps
- Each color map is a 2D array, therefore a color image is a 3D array

j

i

62	79	23	119	120	105	4	0
10	10	9	62	12	78	34	0
10	58	197	46	46	0	0	48
176	135	5	188	191	68	0	49
2	1	1	29	26	37	0	77
0	89	144	147	187	102	62	208
255	252	0	166	123	62	0	31
166	63	127	17	1	0	99	30

- Multispectral images: may include frequencies beyond visible light spectrum
- Depth maps
  - Depth sensors, Kinect
  - Lidar (light and radar?)
  - Calculated using stereo, motion, shadows, structured light, etc.

An image is a function

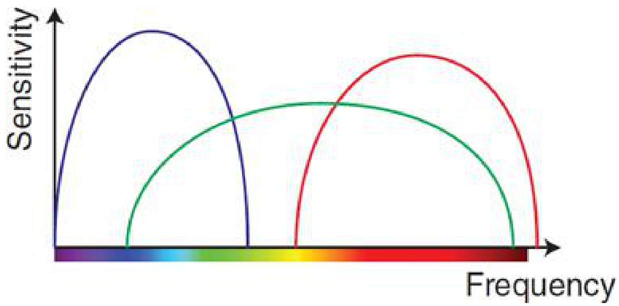
- It can also be thought of as a function, *f*, from R^2 to R:
  - $f(x, y)$ gives the **intensity** at position (x, y)

- A color image is three functions pasted together:

$$f(x, y) = \begin{bmatrix} r(x, y) \\ g(x, y) \\ b(x, y) \end{bmatrix}$$

Color

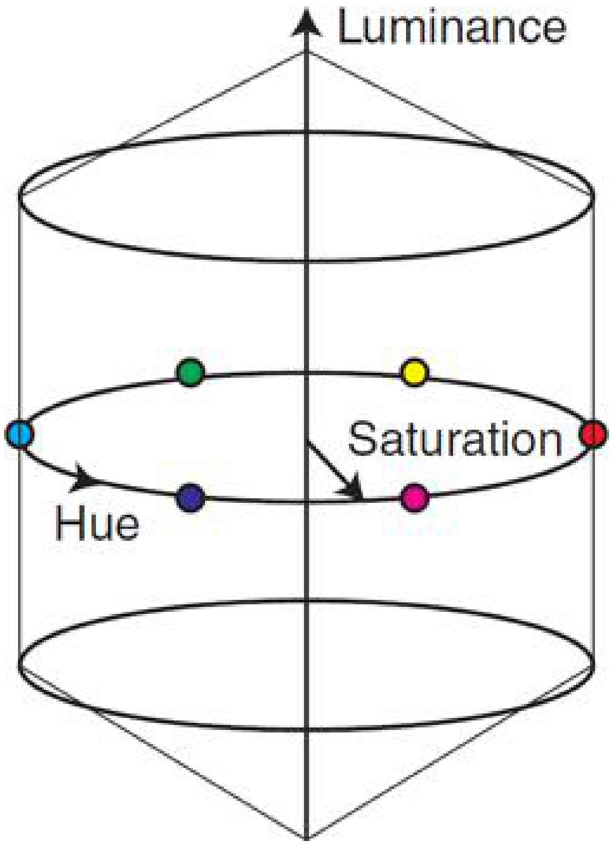
- Human vision system
  - Three different kinds of cones responding to different ranges in the color spectrum
- Digital cameras
  - Red, Green, Blue sensors (sensitive to different portions of the color spectrum)
- Standards
  - CIE RGB
  - CIE XYZ:
    - Y is the luminance or perceived brightness
  - CIELAB (L*a*b*)
    - Based on how human subjects perceive different colors
    - L* is the lightness
  - L*u*v*
  - YCbCr (used in compression algorithms)
  - Many more …



$$\boxed{Y = 0.299R + 0.587G + 0.114B}$$

HLS color images [3]

- Hue/ Luminance/ Saturation



- Skin detection

(Saturation >= 0.2) AND
(0.5 < Luminance/Saturation < 3.0) AND
(Hue <= 28° OR Hue >= 330°)

- Red eye removal

Compression & File formats

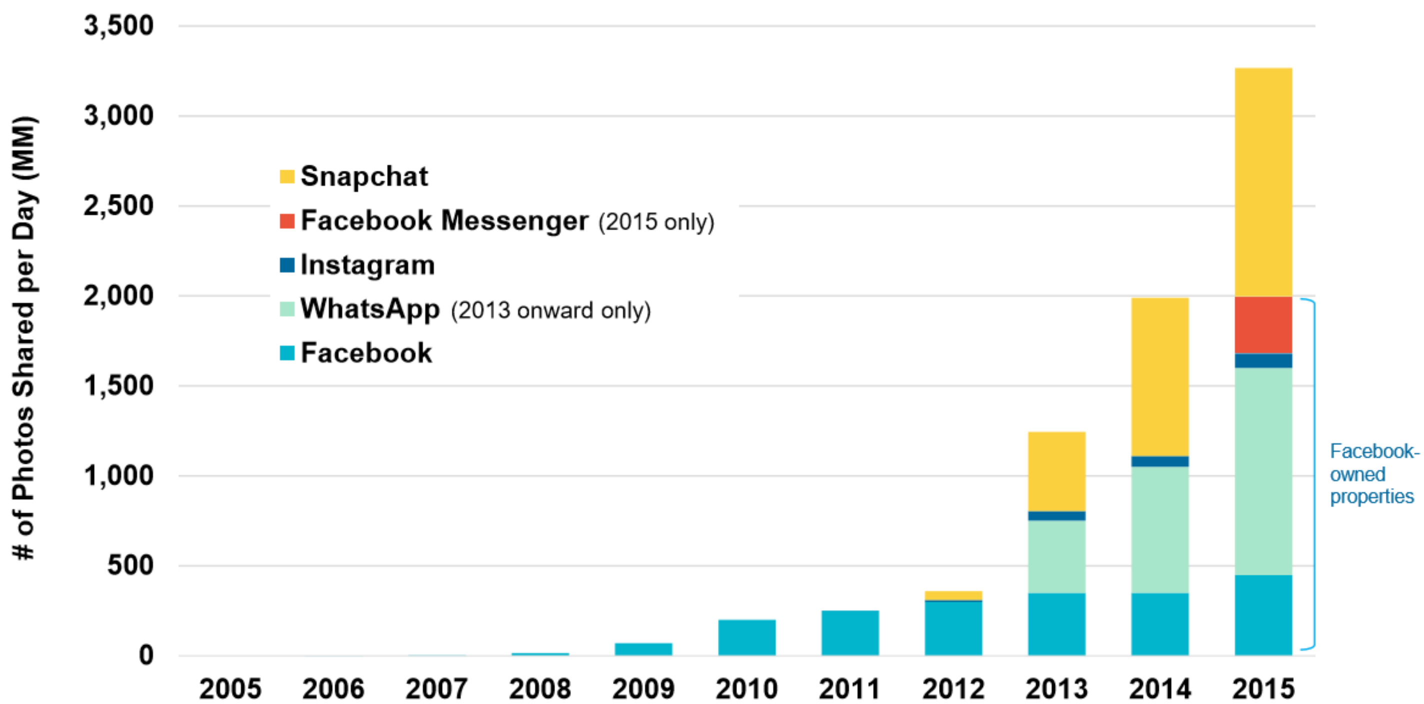
- Lossless
  - RAW
  - PNG
- Lossy
  - JPEG
- Each format has a prescribed method for saving the image information.
  - TIF
  - BMP
  - PPM, PGM
  - …

Need for Compression

- One image, 1440 x 1080, color
  - #pixels: 1440 * 1080 = 1,555,200
  - #bytes: (above) * 3 (1 byte per color channel)= 4,665,600
  - = 4.45 MB
- More than 3×10^9 photos shared on internet daily [http://www.kpcb.com/file/2016-internet-trends-report - page 90]
  - Assuming above resolution:
  - = 4.45 MB * 3 * 10^9
  - = 12,730 TB = 12.43 PB
- A video has about 25 to 50 frames per second
  - 30 seconds of video @ above resolution, 25 fps
  - = 30 * 25 * 4.45 MB = 3.26 TB

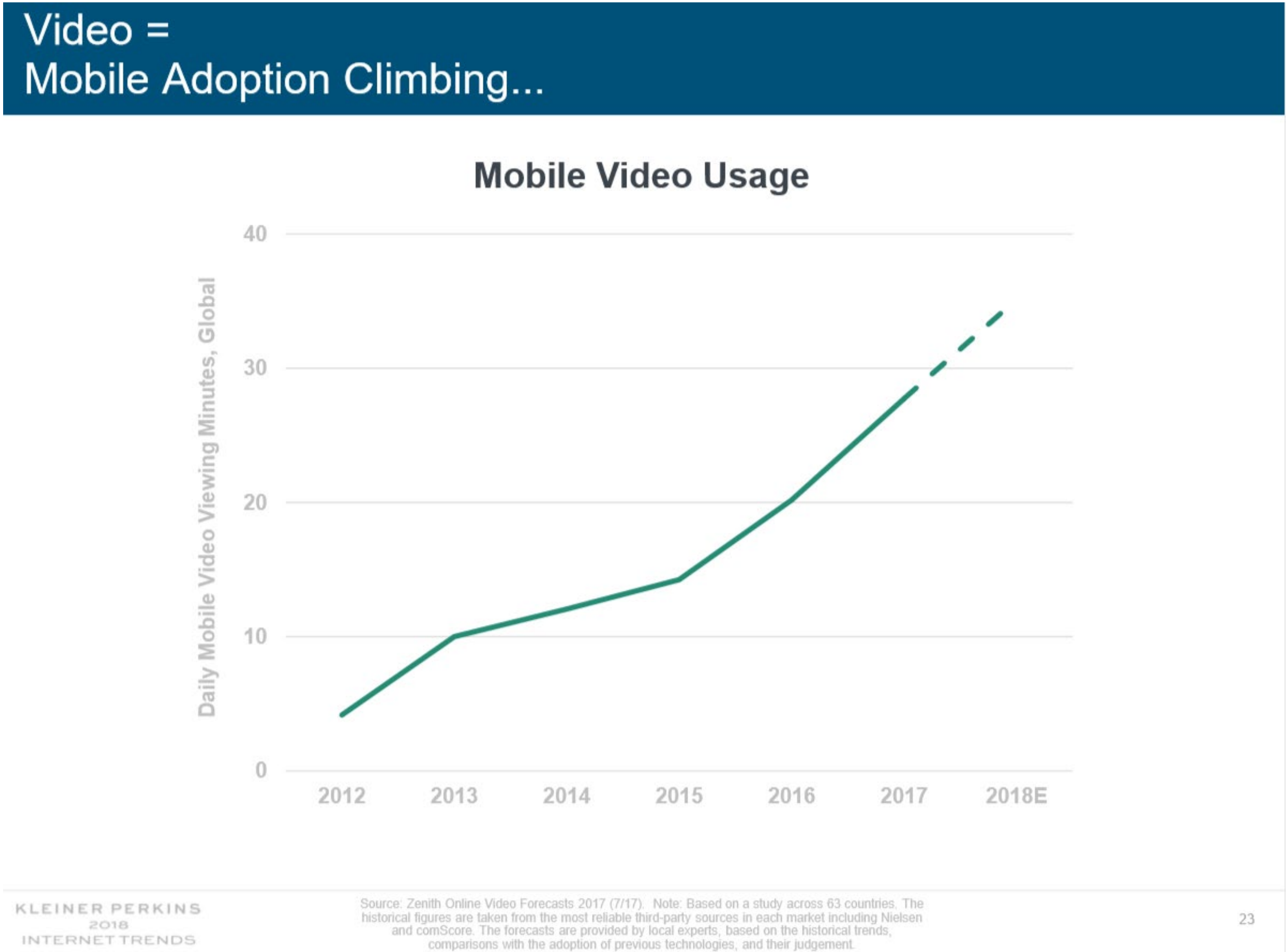
Image Growth Remains Strong

Daily Number of Photos Shared on Select Platforms, Global, 2005 – 2015



Source: Snapchat, Company disclosed information, KPCB estimates
Note: Snapchat data includes images and video. Snapchat stories are a compilation of images and video. WhatsApp data estimated based on average of photos shared disclosed in Q1:15 and Q1:16. Instagram data per Instagram press release. Messenger data per Facebook (~9.5B photos per month). Facebook shares ~2B photos per day across Facebook, Instagram, Messenger, and WhatsApp (2015).

[Source: http://www.kpcb.com/file/2016-internet-trends-report - page 90]

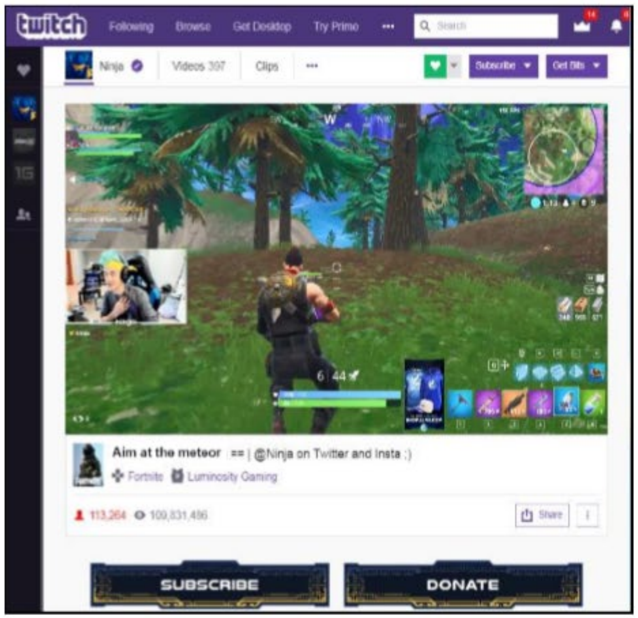


[Source: https://www.kleinerperkins.com/perspectives/internet-trends-report-2018/- page 23]

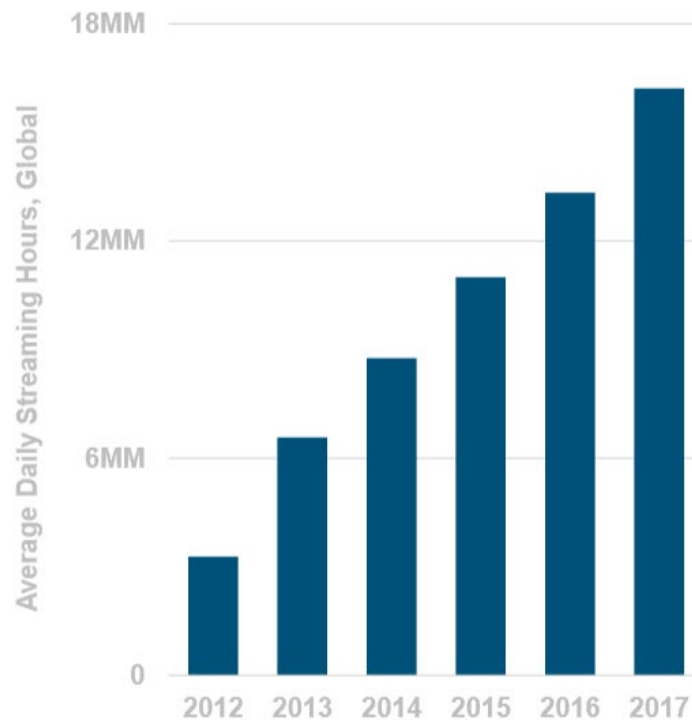
…Video =
New Content Types Emerging

Fortnite Battle Royale

Most Watched Game on Twitch



Twitch Streaming Hours



KLEINER PERKINS
2018
INTERNET TRENDS

Source: Twitch (3/18). Note: Tyler "Ninja" Blevins Twitch stream has 7MM+ followers (#1 ranked) as of 5/29/18 based on Social Blade data.

24

[Source: https://www.kleinerperkins.com/perspectives/internet-trends-report-2018/- page 24]

Image Compression
just an overview

- Conversion to YCbCr
  - Human Visual system has lower sensitivity to color than to luminance changes
  - Lower resolution for color (Cb and Cr), Higher resolution for luminance (Y)
- Discrete Cosine Transform (DCT) applied to blocks (e.g. 8 x 8 blocks) in the image
  - Detecting more frequent values
- Quantization and Huffman Coding
  - Throw away values that have low frequency (lossy compression)
  - Use more bits for describing higher frequency values with higher precision

OpenCV: Image files

OpenCV HighGUI
High-level Graphical User Interface

- Portable
- Three parts:
  - Hardware
    - Concerned with the operation of cameras
  - File System
    - Concerned with loading and saving images
    - Read videos (same way as reading from cameras)
  - GUI
    - Open a window, display an image
    - Respond to mouse and keyboard events
    - Slider bars
- More info: [2] Chapter 8

Loading Images [2]

- cv::imread()
  - Reads image from file
  - Decompresses to an image array

```
cv::Mat cv::imread(
    const string& filename,                  // Input filename
    int flags = cv::IMREAD_COLOR             // Flags set how to interpret file
);
```

- If it fails, returns an empty cv::Mat
- See Table 8.1 for flags

Saving Images [2]

- cv::imwrite()
  - Compresses image to specific format
  - Writes compressed image to file

```
bool cv::imwrite(
    const string& filename,                              // Input filename
    cv::InputArray image,                                // Image to write to file
    const vector<int>& params = vector<int>()    // (Optional) formats
);
```

  - The extension in filename is used to determine format
  - Supported formats: .jpg or .jpeg; .jp2, .tif or .tiff, .png, .bmp, .ppm, .pgm
  - Returns true if successful.

Saving images- cont.

- Second argument is the image to be stored.
- Only 8-bit, single- or three-channel images written by imwrite()
- See Table 8-2 for parameters
- Codecs (<u>co</u>mpression and <u>dec</u>ompression libraries):
  - OpenCV codecs
  - External libraries

Compression and Decompression

- Two more functions that can deal with compressing and decompressing image data:

```
void cv::imencode(
      const string& ext,                         // Extension specifies codec
      cv::InputArray img,                        // Image to be encoded
      vector<uchar>& buf,                        // Encoded file bytes go here
      const vector<int>& params = vector<int>()  // (Optional) formats
);
```

```
cv::Mat cv::imdecode(
      cv::InputArray buf,                        // Encoded file bytes are here
      int flags = cv::IMREAD_COLOR               // Flags set how to interpret file
);
```

Reading Video

- cv::VideoCapture
  1. Reading frames from a video file

```
cv::VideoCapture::VideoCapture (
    const string& filename          // Input filename
);
```

  - If opened successfully, cv::VideoCapture::isOpened() will return true

  2. Reading frames from a camera

```
cv::VideoCapture::VideoCapture (
    int device                      // Video capture device id
);
```

  - Identification number – zero when only one camera
  - Can specify domain (see [2] Chap 8, Table 8-3, pp. 191)

OpenCV: Data Types

OpenCV Data Types (1)

- **Point** class
  - Suitable for 2 or 3 dimensional point coordinates
  - Member data: *x, y, z* (if 3-D)
  - Member functions to calculate dot product, cross product, query if point is *inside* a rectangle
  - Aliases: *cv::Point2i, cv::Point2f, cv::Point2d, cv::Point3i, cv::Point3f, cv::Point3d*

- **Rect** class
  - Members upper left corner (x, y) and width and height
  - Member functions to calculate area, get corners, query if a point is inside

Abbreviations

- The following abbreviations are often used in aliases:

Abbreviation	Data Type
b	unsigned char
w	unsigned short
s	short
i	int
f	float
d	double

OpenCV Data Types (2)

- **Size** class
  - Members width and height
  - Function to calculate area
  - Aliases: *cv::Size, cv::Size2i, cv::Size2f*

- **Scalar** class
  - A four-dimensional point class
  - Example:
  - *cv::Scalar s(x0);*
  - *cv::Scalar s(x0, x1, x2, x3);*

OpenCV Data Types (3)

- Fixed vector (**Vec**) class
  - Suitable for small fixed-size vectors
  - Aliases in form of *cv::Vec{2,3,4,6}{b,s,w,I,f,d}*, e.g. *cv::Vec2s, cv::Vec6f*
  - Accessing members by [] or (), e.g. *v[2]* or *v(2)*
  - Functions for performing cross products
- Fixed matrix (**Matx**) class
  - Suitable for matrices with fixed known dimensions
  - Optimized for small matrices
  - Aliases in form of *cv::Matx{1,2,..}{1,2..}{f,d},* e.g. *cv::Matx33f, cv::Matx43d*
  - Functions for creating zero, unity, or random matrices
  - Functions for performing matrix operations, solving linear systems
  - Accessing members by (), e.g. *m(i,j)*

OpenCV Data Types (4)

- **Range** class
  - A continuous sequence of numbers
  - *Cv::Range rng(0,4)* includes 0,1,2,3
  - Functions: *size(), empty(), all()*

- **Ptr** template
  - Smart pointer, no need to worry about deallocating
  - Example:
  - *cv::Ptr<Matx33f> p (new cv::Matx33f);* or
  - *cv::Ptr<Matx33f> p = makePtr<cv::Matx33f> ( );*

OpenCV Data Types (5)

- cv::**InputArray**, cv::**OutputArray**, cv::**InputOutputArray** classes
  - Typically seen in the definition of library routines
  - A way to include different types in the definition, including:

    cv::Vec, cv::Matx, cv::Scalar, std::vector(), cv::Mat, cv::SparseMat
  - *cv::InputArray* is assumed to be *const*

Large Array Types

- cv::**Mat**
  - An n-dimensional array of single numbers, or
  - An (n-1) dimensional array of vectors (multichannel array)

- cv::**SparseMat**
  - Used for sparse matrices, where the majority of the array elements are zeros

Gray scale image (one color channel)

	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

Color image- BGR color format

	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

Create an array

- A color image

  = a 3-dimensional array of single values

  = a 2-dimensional array of vector values (3 color values)

```
cv::Mat m;

// create data area for 5 rows and 10 columns of 3 color
values, each defined as an 8 bit (unsigned)
m.create( 5, 10, CV_8UC3 );

// set the second channel to 255, others to zero
m.setTo(cv::Vec3b(0, 255, 0)

Or
cv::Mat m ( 5, 10, CV_8UC3, cv::Vec3b(0, 255, 0) );
```

Creating commonly used arrays

Function	What does it do?
cv::Mat::zeros(rows, cols, type);	Creates a cv::Mat of all zeros
cv::Mat::ones(rows, cols, type);	Creates a cv:Mat of all ones for the first channel, and all zeros for the other channels
cv::Mat::eye(rows, cols, type);	Creates a cv:Mat of all zeros, except for pixels on the diagonal (where row# = col#), the first channel will be set to 1. (Identity matrix)

```
Example:
cv::Mat m = cv::Mat::zeros( 5, 10, CV_8UC3);
```

Memory layout

- In a one-dimensional array, the elements are sequential
- In a two-dimensional array, the data is organizes into rows
- Rows of an array may not be absolutely sequential; there may be gaps

Accessing array elements- by location

- Use at<>() template member function
- Specify the type

```
// 2D array of ints
cv::Mat m = cv::Mat::zeros( 5, 10, int );

// To access the element (2,3)
m.at<int>(2,3)

// For example:
printf("The value of element (2,3) is %d \n",
                                    m.at<int>(2,3) );
```

Accessing array elements- by location

```
// 2D array of 3 unsigned char channels (most common)
cv::Mat m ( 5, 10, CV_8UC3, cv::Vec3b(0, 255, 0) );

// To access the 3 channels of pixel (2,3)
m.at<cv::Vec3b>(2,3)[0]
m.at<cv::Vec3b>(2,3)[1]
m.at<cv::Vec3b>(2,3)[2]


// 2D array of 3 float channels
cv::Mat m ( 5, 10, CV_32FC3, cv::Vec3f(0.0f, 0.0f, 1.0f) );

// To access the 3 channels of pixel (2,3)
m.at<cv::Vec3f>(2,3)[0]
m.at<cv::Vec3f>(2,3)[1]
m.at<cv::Vec3f>(2,3)[2]
```

Example: max 'red' value

```
// 2D array of 3 unsigned char channels
cv::Mat m(5, 10, CV_8UC3);
// populate with random values between 0 to 255
cv::randu(m, 0, 255);

cv::imshow("Random image", m);

cv::Size sz = m.size();
short  max = 0;
for (int i = 0; i < sz.height; i++) {
    for (int j = 0; j < sz.width; j++) {
        if (m.at<cv::Vec3b>(i, j)[2] > max) {
            max = (short)m.at<cv::Vec3b>(i, j)[2];
        }
    }
}
cout << "Maximum red value: " << max << endl;
```

Accessing rows of an array

- Use ptr<>() template member function
- Specify the type

```
// 2D array of 3 unsigned char channels
cv::Mat m ( 5, 10, CV_8UC3, cv::Vec3b(0, 255, 0) );

// To get a pointer pointing to the 1st channel of 1st pixel
in row 2
Unsigned char *p = m.ptr<cv::Vec3b>(2) ;
```

Accessing array elements- using an iterator

- Cv::MatIterator_<> and cv::MatConstIterator_<>

```
// 2D array of 3 float channels
cv::Mat m = cv::Mat::eye( 5, 10, CV_32FC3 );


total = 0.0f;
cv::MatConstIterator_<cv::Vec3f> it = m.begin<cv::Vec3f>();
while (it != m.end<cv::Vec3f>()) {
    total += (*it)[0];
    it++;
}
```

Matrix operations (1)

Example	Description
m1 + m2 m1 – m2	Addition of matrices Subtraction of matrices
m1 + s or s + m1 m1 – s or s – m1	Addition or subtraction of a matrix and a single value
-m1	Negation of a matrix (applied to all elements)
s * m1 or m1 * s	Multiplying (all elements of) a matrix by a single value
m0.mul(m1)	Per-element multiplication of two matrices
m0 / m1	Per-element division of two matrices
m0 * m1	Matrix multiplication
m1.inv(method)	Matrix inversion (default method: DECOMP_LU)
m1.t()	Matrix transpose

Matrix operations (2)

<table>
<tr><th>Example</th><th>Description</th></tr>
<tr><td>m1>m2, m1>=m2,
m1<m2, m1<=m2,
m1==m2</td><td>Per-element comparison, a uchar matrix is returned with elements equal to 0 or 255</td></tr>
<tr><td>m0&m1, m0|m1,
m0^m1, ~m0
m0&s, s&m0, m0|s,
s|m0, m0^s, s^m0</td><td>Bitwise logical operations</td></tr>
<tr><td>min(m0,m1);
max(m0,m1);</td><td>Per-element min and max</td></tr>
<tr><td>min(m0,s) or min(s,m0)
max(m0,s) or max(s,m0)</td><td>Min and Max between elements of a matrix and a single value</td></tr>
<tr><td>cv::abs(m0)</td><td>Per-element absolute value</td></tr>
<tr><td>m0.dot(m1);</td><td>Dot product</td></tr>
<tr><td>m0.cross(m1);</td><td>Cross product (defined only for 3 x 1 matrices)</td></tr>
</table>

References

[1] Computer Vision: Algorithms and Applications, R. Szeliski (http://szeliski.org/Book)

[2] Learning OpenCV 3, A. Kaehler & G. Bradski

- Available online through Safari Books, Seneca libraries
- https://senecacollege-primo.hosted.exlibrisgroup.com/primo-explore/fulldisplay?docid=01SENC_ALMA5153244920003226&context=L&vid=01SENC&search_scope=default_scope&tab=default_tab&lang=en_US

[3] Practical introduction to Computer Vision with OpenCV, Kenneth Dawson-Howe

- Available through Seneca libraries
- https://senecacollege-primo.hosted.exlibrisgroup.com/primo-explore/fulldisplay?docid=01SENC_ALMA5142810950003226&context=L&vid=01SENC&search_scope=default_scope&tab=default_tab&lang=en_US