Camera Processing Pipeline

Kari Pulli
VP Computational Imaging Light
Imaging without optics?

- Each point on sensor
  - would record the integral of light
  - arriving from every point on subject

- All sensor points would record similar colors
Pinhole camera (a.k.a. camera obscura)

Linear perspective with viewpoint at pinhole
Effect of pinhole size

Photograph made with small pinhole

Photograph made with larger pinhole
Stopping down the pinhole

- Large pinhole
  - geometric blur

- Optimal pinhole
  - too little light

- Small pinhole
  - diffraction blur
Add a lens to get more light
Real lenses are complex
Thin lens approximation: Gauss’s ray diagram

http://graphics.stanford.edu/courses/cs178/applets/thinlens.html
Changing the focus distance

To focus on objects at different distances:
- move sensor relative to the lens

\[ \frac{1}{s_o} + \frac{1}{s_i} = \frac{1}{f} \]
Changing the focus distance

To focus on objects at different distances

- move sensor relative to the lens

At \( s_o = s_i = 2f \) we get 1:1 imaging (macro) because

\[
\frac{1}{s_o} + \frac{1}{s_i} = \frac{1}{f}
\]

\[
\frac{1}{2f} + \frac{1}{2f} = \frac{1}{f}
\]
Changing the focus distance

To focus on objects at different distances
- move sensor relative to the lens

At $s_o = s_i = 2f$ we get 1:1 imaging (macro) because

$$\frac{1}{s_o} + \frac{1}{s_i} = \frac{1}{f}$$

Can’t focus on objects closer to the lens than $f$

http://graphics.stanford.edu/courses/cs178/applets/gaussian.html
Circle of confusion
• Depth of field (DOF) = the range of distances that are in focus

http://graphics.stanford.edu/courses/cs178/applets/dof.html
Chromatic aberration

- Different wavelengths refract at different rates
  - so have different focal lengths
- Correct with achromatic doublet
  - strong positive lens + weak negative lens
  - = weak positive compound lens
  - align red and blue
Lens distortion

- Radial change in magnification
  - (a) pincushion
  - (b) barrel distortion
Vignetting

- Irradiance is proportional to
  - projected area of aperture as seen from pixel
  - projected area of pixel as seen from aperture
  - distance$^2$ from aperture to pixel

Combining all these
- each $\sim$ a factor of $\cos \theta$
- light drops as $\cos^4 \theta$

- Fix by calibrating
  - take a photo of a uniformly white object
  - the picture shows the attenuation, divide the pixel values by it
CMOS sensor

Anatomy of the Active Pixel Sensor Photodiode

- Microlens
- Red Color Filter
- Reset Transistor
- Row Select Bus
- Amplifier Transistor
- Column Bus Transistor
- Photodiode
- Silicon Substrate
- Potential Well
- Analog Signal Processing
- Analog-to-Digital Conversion
- Bayer Mosaic Filters
- Active Pixel Sensor Color Imaging Array
- CMOS Image Sensor Die
- Image Sensor Die
- Pad Ring
- Digital Logic (Interface, Timing, Processing, Output)
- Clock and Timing Control
Front- vs. Back-illuminated sensor
Anti-aliasing filter

- Two layers of birefringent material
  - splits one ray into 4 rays

anti-aliasing filter removed normal
From “raw-raw” to RAW

- **Pixel Non-Uniformity**
  - each pixel has a slightly different sensitivity to light
    - typically within 1% to 2%
  - reduce by calibrating an image with a flat-field image
    - also eliminate the effects of vignetting and other optical variations

- **Stuck pixels**
  - some pixels are turned always on or off
  - identify, replace with filtered values

- **Dark floor**
  - temperature adds noise
  - sensors usually have a ring of covered pixels around the exposed sensor, subtract their signal
AD Conversion

- Sensor converts the continuous light signal to a continuous electrical signal
- The analog signal is converted to a digital signal
  - at least 10 bits (even on cell phones), often 12 or more
  - (roughly) linear sensor response
ISO = amplification in AD conversion

Before conversion the signal can be amplified
- ISO 100 means no amplification
- ISO 1600 means 16x amplification
+: can see details in dark areas better
-: noise is amplified as well; sensor more likely to saturate
From “raw-raw” to RAW

- **Pixel Non-Uniformity**
  - Each pixel in a CCD has a slightly different sensitivity to light, typically within 1% to 2% of the average signal.
  - Can be reduced by calibrating an image with a flat-field image.
  - Flat-field images are also used to eliminate the effects of vignetting and other optical variations.

- **Stuck pixels**
  - Some pixels are turned always on or off.
  - Identify, replace with filtered values.

- **Dark floor**
  - Temperature adds noise.
  - Sensors usually have a ring of covered pixels around the exposed sensor, subtract their signal.
Color filter array

- Bayer pattern
## Demosaicking

<table>
<thead>
<tr>
<th>Original Scene (shown at 200%)</th>
<th>What Your Camera Sees (through a Bayer array)</th>
</tr>
</thead>
</table>

The image on the left represents the original scene of a detailed architectural feature, shown at double its original size. The image on the right illustrates what a camera captures when equipped with a Bayer array, which is a common color-interleaving pattern for capturing color images with a single-chip color sensor. The Bayer array captures different color components in a repeating pattern, leading to a mosaic-like appearance compared to the original scene.
Your eyes do it too...
Demosaicking
First choice: bilinear interpolation

- Easy to implement
- But fails at sharp edges
Take edges into account

- Use bilateral filtering
  - avoid interpolating across edges

ADAPTIVE DEMOSAICKING
Ramanath, Snyder, JEI 2003

**Fig. 3** Bilateral filtering: (a) original image, (b) image corrupted by Gaussian noise, (c) 7×7 blur kernel, (d) 7×7 similarity kernel at row=18, col=18, (e) 7×7 bilateral filter kernel, and (f) resulting image (denoised and sharpened).
Take edges into account

- Predict edges and adjust assumptions
  - luminance correlates with RGB
  - edges = luminance change

- When estimating G at R
  - if the R differs from bilinearly estimated R
    - \( \rightarrow \) luminance changes

- Correct the bilinear estimate
  - by the difference between the estimate and real value

\[
\hat{G}(i, j) = \hat{G}(i, j) + \alpha \Delta \hat{R}(i, j)
\]

Figure 2. Filter coefficients for our proposed linear interpolation scheme.
Denoising using non-local means

- Most image details occur repeatedly
- Each color indicates a group of squares in the image which are almost indistinguishable
- Image self-similarity can be used to eliminate noise
  - it suffices to average the squares which resemble each other

Image and movie denoising by nonlocal means
Buades, Coll, Morel, IJCV 2006
BM3D (Block Matching 3D)

Image denoising by sparse 3D transform-domain collaborative filtering

Kostadin Dabov, Alessandro Foi, Vladimir Katkovnik, and Karen Egiazarian, Senior Member, IEEE
The CIE XYZ System

- A standard created in 1931 by CIE
  - Commission Internationale de L'Eclairage
- Defined in terms of three color matching functions
- Given an emission spectrum, we can use the CIE matching functions to obtain the $x$, $y$ and $z$ coordinates
  - $y$ corresponds to luminance perception

http://graphics.stanford.edu/courses/cs178/applets/threedgamut.html
Intensity is measured as the distance from origin
- black = (0, 0, 0)

Chromaticity coordinates give a notion of color independent of brightness

A projection of the plane $x + y + z = 1$ yields a chromaticity value dependent on
- dominant wavelength (= hue), and
- excitation purity (= saturation)
  - the distance from the white at $(1/3, 1/3, 1/33)$
Perceptual (non-)uniformity

- The XYZ color space is not perceptually uniform!
- Enlarged ellipses of constant color in XYZ space
CIE L* a* b*: uniform color space

- Lab is designed to approximate human vision
  - it aspires to perceptual uniformity
  - L component closely matches human perception of lightness

- A good color space for image processing
Break RGB to Lab channels
Blur “a” channel (red-green)
Blur “b” channel (blue-yellow)
Blur “L” channel
Family of color spaces for video encoding
- Uses the fact that eye is sensitive to luminance
- Filters colors down (2:1, 4:1)

Channels
- Y = luminance [linear]; Y’ = luma [gamma corrected]
- CbCr / UV = chrominance [always linear]

Y’CbCr is not an absolute color space
- it is a way of encoding RGB information
- the actual color depends on the RGB primaries used
How many bits are needed for smooth shading?

With a given adaptation, human vision has contrast sensitivity ~1%
- call black 1, white 100
- you can see differences
  - 1, 1.01, 1.02, … needed step size ~ 0.01
  - 98, 99, 100 needed step size ~ 1
- with linear encoding
  - delta 0.01
    - 100 steps between 99 & 100 → wasteful
  - delta 1
    - only 1 step between 1 & 2 → lose detail in shadows
- instead, apply a non-linear power function, gamma
  - provides adaptive step size
Gamma encoding

With the “delta” ratio of 1.01
- need about 480 steps to reach 100
- takes almost 9 bits

8 bits, nonlinearly encoded
- sufficient for broadcast quality digital TV
- contrast ratio ~ 50 : 1

With poor viewing conditions or display quality
- fewer bits needed

http://graphics.stanford.edu/courses/cs178/applets/gamma.html
Luminance from RGB

If three sources of same radiance appear R, G, B:
- **green** will appear the brightest, it has high luminous efficiency
- **red** will appear less bright
- **blue** will be the darkest

Luminance by NTSC: \[0.2990 \, R + 0.5870 \, G + 0.1140 \, B\]
- based on phosphors in use in 1953

Luminance by CIE: \[0.2126 \, R + 0.7152 \, G + 0.0722 \, B\]
- based on contemporary phosphors

Luminance by ITU: \[0.2125 \, R + 0.7154 \, G + 0.0721 \, B\]

\[\frac{1}{4} \, R + \frac{5}{8} \, G + \frac{1}{8} \, B\] works fine
- quick to compute: \(R\gg2 + G\gg1 + G\gg3 + B\gg3\)
- range is [0, 252]
Cameras use sRGB

- sRGB is a standard RGB color space (since 1996)
  - uses the same primaries as used in studio monitors and HDTV
  - and a gamma curve typical of CRTs
  - allows direct display

- First need to map from sensor RGB to standard
  - need calibration
sRGB from XYZ

Linear relation between XYZ and sRGB:

\[
\begin{pmatrix}
X \\
Y \\
Z
\end{pmatrix} =
\begin{pmatrix}
0.4124 & 0.3576 & 0.1805 \\
0.2126 & 0.7152 & 0.0722 \\
0.193 & 0.1192 & 0.9505
\end{pmatrix}
\begin{pmatrix}
R_{sRGB} \\
G_{sRGB} \\
B_{sRGB}
\end{pmatrix}
\]

Primaries according to ITU-R BT.709.3

RGB to sRGB:

\[
R'_{sRGB} = \begin{cases}
12.92 R_{sRGB} & \text{if } R_{sRGB} \leq 0.0031308 \\
1.055 (R_{sRGB}^{1/2.4}) - 0.055 & \text{otherwise}
\end{cases}
\]

RGB to 8Bit:

\[
R_{8Bit} = \text{round}[255 R'_{sRGB}]
\]
Image processing in linear or non-linear space?

Simulating physical world
- use linear light
- a weighted average of gamma-corrected pixel values is not a linear convolution!
  - Bad for antialiasing
- want to numerically simulate lens?
  - Undo gamma first

Dealing with human perception
- using non-linear coding allows minimizing perceptual errors due to quantization
Film response curve

- **Toe region**
  - the chemical process is just starting

- **Middle**
  - follows a power function
  - if a given amount of light turned half of the grain crystals to silver, the same amount more turns half of the rest

- **Shoulder region**
  - close to saturation

- **Toe and shoulder** preserve more dynamic range at dark and bright areas
  - at the cost of reduced contrast

- **Film has more dynamic range than print**
  - ~12bits
Digital camera response curve

Digital cameras modify the response curve

May use different response curves at different exposures
impossible to calibrate and invert!
Automated selection of key camera control values

- auto-focus
- auto-exposure
- auto-white-balance
Contrast-based auto-focus

ISP can filter pixels with configurable IIR filters to produce a low-resolution sharpness map of the image.

The sharpness map helps estimate the best lens position by summing the sharpness values (= Focus Value)
  - either over the entire image
  - or over a rectangular area

http://graphics.stanford.edu/courses/cs178/applets/autofocusCD.html
Hunt for the peak

- First coarse search
  - back up after peak, do a finer search
Auto-White-Balance

- The dominant light source *(illuminant)* produces a color cast that affects the appearance of the scene objects.
- The color of the illuminant determines the color normally associated with white by the human visual system.
- Auto-white-balance
  - Identify the illuminant color
  - Neutralize the color of the illuminant

(source: www.cambridgeincolour.com)
Identify the color of the illuminant

- Prior knowledge about the ambient light
  - Candle flame light (1850°K)
  - Sunset light (2000°K)
  - Summer sunlight at noon (5400°K)
  - ...

- Known reference object in the picture
  - best: find something that is white or gray

- Assumptions about the scene
  - Gray world assumption
    (gray in sRGB space!)
Best way to do white balance

- Grey card
  - take a picture of a neutral object (white or gray)
  - deduce the weight of each channel

- If the object is recoded as $r_w$, $g_w$, $b_w$
  - use weights $k/r_w$, $k/g_w$, $k/b_w$
    - where $k$ controls the exposure
Brightest pixel assumption

- Highlights usually have the color of the light source
  - at least for dielectric materials

- White balance by using the brightest pixels
  - plus potentially a bunch of heuristics
  - in particular use a pixel that is not saturated / clipped
- Colors of a black-body heated at different temperatures fall on a curve (Planckian locus)
- Colors change non-linearly with temperature
  - but almost linearly with reciprocal temperatures $1/T$
Mapping the colors

For a given sensor
- pre-compute the transformation matrices between the sensor color space and sRGB at different temperatures

Estimate a new transformation
- by interpolating between pre-computed matrices
- ISP can apply the linear transformation
Estimating the color temperature

- Use scene mode
- Use gray world assumption (R = G = B) in sRGB space
  - really, just R = B, ignore G
- Estimate color temperature in a given image
  - apply pre-computed matrix to get sRGB for $T_1$ and $T_2$
  - calculate the average values R, B
  - solve $\alpha$, use to interpolate matrices (or 1/T)

\[
\frac{1}{T} = (1 - \alpha) \frac{1}{T_1} + \alpha \frac{1}{T_2}
\]

\[
R = (1 - \alpha)R_1 + \alpha R_2, \quad B = (1 - \alpha)B_1 + \alpha B_2
\]
Auto-exposure

Goal: well-exposed image (not a very well defined goal!)

Possible parameters to adjust

- Exposure time
  - Longer exposure time leads to brighter image, but also motion blur
- Aperture (f-number)
  - Larger aperture (smaller f-number) lets more light in causing the image to be brighter, also makes depth of field shallower
  - Phone cameras often have fixed aperture
- Analog and digital gain
  - Higher gain makes image brighter but amplifies noise as well
- ND filters on some cameras
Exposure metering

- Cumulative Density Function of image intensity values
  - $P$ percent of image pixels have an intensity lower than $Y$
Exposure metering examples

**Adjustment examples**

- $P = 0.995, \ Y = 0.9$
  - max 0.5% of pixels are saturated (highlights)
- $P = 0.1, \ Y = 0.1$
  - max 10% of pixels are under-exposed (shadows)
- Auto-exposure somewhere in between, e.g., $P = 0.9, \ Y = 0.4$

---

![Highlights](image1)

![Auto-exposure](image2)

![Shadows](image3)
JPEG Encoding

1. Transform RGB to YUV or YIQ and subsample color
2. DCT on 8x8 image blocks
3. Quantization
4. Zig-zag ordering and run-length encoding
5. Entropy coding
Alternatives?

- **JPEG 2000**
  - ISO, 2000
  - better compression, inherently hierarchical, random access, ...
  - but much more complex than JPEG

- **JPEG XR**
  - Microsoft, 2006; ISO / ITU-T, 2010
  - good compression, supports tiling (random access without having to decode whole image), better color accuracy (incl. HDR), transparency, compressed domain editing

- But JPEG stays
  - too large an install base
Traditional camera APIs

- Real image sensors are pipelined
  - while one frame exposing
  - next one is being prepared
  - previous one is being read out

- Viewfinding / video mode:
  - pipelined, high frame rate
  - settings changes take effect sometime later

- Still capture mode:
  - need to know which parameters were used
  - \(\rightarrow\) reset pipeline between shots \(\rightarrow\) slow

Diagram:

1. Exposure, Frame rate
2. Gain, Digital Zoom
3. Readout
4. Format
5. Demosaic
6. Color corr
...
The FCam Architecture

- A software architecture for programmable cameras
  - that attempts to expose the maximum device capabilities
  - while remaining easy to program
Sensor

- A pipeline that converts requests into images
- No global state
  - state travels in the requests through the pipeline
  - all parameters packed into the requests
Image Signal Processor (ISP)

- Receives sensor data, and optionally transforms it
  - untransformed raw data must also be available
- Computes helpful statistics
  - histograms, sharpness maps
Devices

- Devices (like the Lens and Flash) can schedule Actions to be triggered at a given time into an exposure.
- Tag returned images with metadata.
Everything is visible

- Programmer has full control over sensor settings and access to the supplemental statistics from ISP
- No hidden daemon running autofocus/metering
  - nobody changes the settings under you
FCam: An API for controlling computational cameras.

The FCam API provides mechanisms to control various components of a camera to facilitate complex photographic applications.

To use the FCam, you pass Shots to a Sensor which asynchronously returns Frames. A Shot completely specifies the capture and post-processing parameters of a single photograph, and a Frame contains the resulting image, along with supplemental hardware-generated statistics like a Histogram and SharpnessMap. You can tell Devices (like Lenses or Flashes) to schedule Actions (like firing the flash) to occur at some number of microseconds into a Shot. If timing is unimportant, you can also just tell Devices to do their thing directly from your code. In either case, Devices add tags to returned Frames (like the position of the Lens for that Shot). Tags are key-value pairs, where the key is a string like "focus" and the value is a TagValue, which can represent one of a number of types.
Android Camera Architecture

HAL v3

Camera2 API Core Operation Model
1 Request → 1 image captured → 1 Result metadata + N image buffers

Application Framework
→ Request Queue
→ Camera Apps

Camera HAL Implementation
→ Frame Metadata Queue
→ Image Buffers

CameraDevice
→ capture()

Pending request queue

In-flight capture queue

Output image queues
→ onCaptureComplete()