Camera Processing Pipeline

Kari Pulli
Senior Director
NVIDIA Research
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
Stopping down the pinhole

- Large pinhole
  - geometric blur

- Optimal pinhole
  - too little light

- Small pinhole
  - diffraction blur
Add a lens to get more light
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 because

$$\frac{1}{2f} + \frac{1}{2f} = \frac{1}{f}$$
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 because
  \[ \frac{1}{2f} + \frac{1}{2f} = \frac{1}{f} \]

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

http://graphics.stanford.edu/courses/cs178/applets/gaussian.html
Circle of confusion
Focusing

- Depth of field (DOF) = the range of distances that are in focus
- Diopters [1/m] are used as the units for focal distance
- Focus limits
  - Near focus: the closest distance the device can focus, about 5cm (20 D) in N900
  - Far focus: infinity (0 diopters)

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
Front- vs. Back-illuminated sensor
Anti-aliasing filter

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

Original Scene (shown at 200%)  What Your Camera Sees (through a Bayer array)
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) 7x7 blur kernel, (d) 7x7 similarity kernel at row=18, col=18, (e) 7x7 bilateral filter kernel, and (f) resulting image (denoised and sharpened).
Start with Gaussian filtering

- Here, input is a step function + noise

\[ J = f \otimes I \]
Start with Gaussian filtering

- Spatial Gaussian filter

\[ J = f \odot I \]
Start with Gaussian filtering

Output is blurred

\[ J = f \otimes I \]
The problem of edges

- Weight $f(x, \xi)$ depends on distance from $\xi$ to $x$
- Here, $I(\xi)$ “pollutes” our estimate $J(x)$ at $I(x)$
- It is too different

\[ J(x) = \sum_{\xi} f(x, \xi) I(\xi) \]
Principle of Bilateral filtering

[Tomasi and Manduchi 1998]

- **Penalty** $g$ on the intensity difference

\[
J(x) = \frac{1}{k(x)} \sum_{\xi} f(x, \xi) \left( g(I(\xi) - I(x)) \right) I(\xi)
\]
Bilateral filtering

[Tomasi and Manduchi 1998]

Spatial Gaussian f

\[ J(x) = \frac{1}{k(x)} \sum_{\xi} f(x, \xi) \frac{g(I(\xi) - I(x))}{I(\xi)} \]

output

input
Bilateral filtering

[Tomasi and Manduchi 1998]

- Spatial Gaussian \( f \)
- Gaussian \( g \) on the intensity difference

\[
J(x) = \frac{1}{k(x)} \sum_{\xi} f(x, \xi) \ g(I(\xi) - I(x)) \ I(\xi)
\]
Normalization factor

\[ k(x) = \sum_{\xi} f(x, \xi) g(I(\xi) - I(x)) \]

\[ J(x) = \frac{1}{k(x)} \sum_{\xi} f(x, \xi) g(I(\xi) - I(x)) I(\xi) \]
Bilateral filtering is non-linear

[Tomasi and Manduchi 1998]

- The weights are different for each output pixel

\[ J(x) = \frac{1}{k(x)} \sum_{\xi} f(x, \xi) \quad g(I(\xi) - I(x)) \quad I(\xi) \]
Other view

- The bilateral filter uses the 3D distance
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) $7 \times 7$ blur kernel, (d) $7 \times 7$ similarity kernel at row=18, col=18, (e) $7 \times 7$ 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
    - luminance changes

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

\[ \hat{g}(i, j) = \hat{g}_B(i, j) + \alpha \Delta_R(i, j) \]

**Figure 2.** Filter coefficients for our proposed linear interpolation for de-mosaicing of Bayer-patterned color images. Malvar, He, Cutler, ICASSP 2004
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)

Noisy image

Step 1
Block-wise estimates → Aggregation
Inverse 3D transform
Hard-thresholding
3D transform

Grouping by block-matching

Basic estimate

Step 2
Block-wise estimates → Aggregation → Final Wiener estimate
Inverse 3D transform
Wiener filtering
3D transform

Weight
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
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
YUV, YCbCr, ...

- Family of color spaces for video encoding
  - including in FCam, video and viewfinder usually YUV

- Channels
  - $Y = \text{luminance [linear]}$; $Y' = \text{luma [gamma corrected]}$
  - $\text{CbCr / UV = chrominance [always linear]}$

- 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

- Colors are often filtered down
  - 2:1, 4:1

- Many formulas!
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
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: $0.2990 \times R + 0.5870 \times G + 0.1140 \times B$
- based on phosphors in use in 1953

Luminance by CIE: $0.2126 \times R + 0.7152 \times G + 0.0722 \times B$
- based on contemporary phosphors

Luminance by ITU: $0.2125 \times R + 0.7154 \times G + 0.0721 \times B$

$1/4 \times R + 5/8 \times G + 1/8 \times B$ 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

- The sRGB gamma
  - cannot be expressed as a single numerical value
  - the overall gamma is approximately 2.2, consisting of
    - a linear (gamma 1.0) section near black,
    - and a non-linear section elsewhere
      involving a 2.4 exponent

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

linear relation between XYZ und sRGB:

\[
\begin{bmatrix}
X \\
Y \\
Z
\end{bmatrix}
= 
\begin{bmatrix}
0.4124 \\
0.2126 \\
0.0193
\end{bmatrix}
\begin{bmatrix}
0.3576 \\
0.7152 \\
0.1192
\end{bmatrix}
\begin{bmatrix}
0.1805 \\
0.0722 \\
0.9505
\end{bmatrix}
\]

Primaries according to ITU-R BT.709.3

RGB\_sRGB = \begin{bmatrix}
R \\
G \\
B
\end{bmatrix}

RGB\_sRGB

\[
\begin{align*}
R\_sRGB < 0.0031308 & \quad \Rightarrow \quad R\_'sRGB = 12.92 \times R\_sRGB \\
R\_sRGB > 0.0031308 & \quad \Rightarrow \quad R\_'sRGB = 1.055 \times \left( R\_sRGB^{1/2.4} \right) - 0.055
\end{align*}
\]

RGB\_8Bit = round[255 \times 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

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

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

- **Shoulder region**
  - close to saturation

- **Film has more dynamic range than print**
  - ~12bits

The ‘geography’ of the characteristic curve of a negative material
Digital camera response curve

- Digital cameras modify the response curve

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

- 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
Digital auto-focus (as in FCam)

- Passive autofocus method using contrast measurements
- 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
Auto-focus in FCam

- A history of sharpness values at different lens positions
- FCam provides a helper class called AutoFocus
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°FK)
  - Sunset light (2000°FK)
  - Summer sunlight at noon (5400°FK)
  - ...

- 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
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.
Color temperature

- 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
- FCam provides two precomputed transformations
  - for 3200°K and 7000°K

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., $P = 0.9, \ Y = 0.4$
void meter (Shot s, Frame frame, float P, float Y, float sm) {
    const Histogram &h = frame.histogram();
    int N = h.buckets(); // number of histogram bins

    ... Calculate the cumulative intensity histogram CDF
    ... Determine the histogram bin i, s.t. CDF[i]<=P<CDF[i+1]
    float Y_{curr} = (i+1)/N;
    float adjustment = Y / Y_{curr};
    // Current exposure
    float currExp = frame.exposure * frame.gain;
    float desiredExp = adjustment * currExp ;
    // Make the change smooth
    desiredExp = (1-sm) * desiredExp + sm * currExp;
    ... Set s.exposure and s.gain to fit desiredExposure
}
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
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 (like the Lens and Flash) can
- schedule Actions	o 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
Simple HDR Burst

#include <FCam/Tegra.h>

...
#include <FCam/Tegra.h>

... 

Sensor sensor;
Shot shortReq, midReq, longReq;
Frame short, mid, long;
#include <FCam/Tegra.h>
...
Sensor sensor;
Shot shortReq, midReq, longReq;
Frame short, mid, long:

shortReq.exposure = 10000; // microseconds
midReq.exposure = 40000;
longReq.exposure = 160000;
shortReq.image = Image(sensor.maxImageSize(), RAW);
midReq.image = Image(sensor.maxImageSize(), RAW);
longReq.image = Image(sensor.maxImageSize(), RAW);
#include <FCam/Tegra.h>
...
Sensor sensor;
Shot shortReq, midReq, longReq;
Frame short, mid, long;

shortReq.exposure = 10000; // microseconds
midReq.exposure = 40000;
longReq.exposure = 160000;
shortReq.image = Image(sensor.maxImageSize(), RAW);
midReq.image = Image(sensor.maxImageSize(), RAW);
longReq.image = Image(sensor.maxImageSize(), RAW);

sensor.capture(shortReq);
sensor.capture(midReq);
sensor.capture(longReq);
Simple HDR Burst

#include <FCam/Tegra.h>
...
Sensor sensor;
Shot shortReq, midReq, longReq;
Frame short, mid, long:

shortReq.exposure = 10000; // microseconds
midReq.exposure = 40000;
longReq.exposure = 160000;
shortReq.image = Image(sensor.maxImageSize(), RAW);
midReq.image = Image(sensor.maxImageSize(), RAW);
longReq.image = Image(sensor.maxImageSize(), RAW);

sensor.capture(shortReq);
sensor.capture(midReq);
sensor.capture(longReq);
short = sensor.getFrame();
mid = sensor.getFrame();
long = sensor.getFrame();
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.
Shot specifies capture & post-process

- **Sensor parameters**
  - analog gain (≅ ISO)
  - exposure time (in microseconds)
  - total time (to set frame rate)
  - output resolution
  - format (raw or demosaicked [RGB, YUV])
  - white balance (only relevant if format is demosaicked)
  - memory location where to place the Image data
  - unique id (auto-generated on construction)

- **Configures fixed-function statistics**
  - region for Histogram
  - region and resolution for Sharpness Map
A Shot is passed to a Sensor

Sensor manages a Shot queue in a separate thread
- Sensor::capture()
  - just sticks a Shot on the end of the queue
- Sensor::stream()
  - adds a copy of Shot to queue when the queue becomes empty

Change the parameters of a streaming Shot
- just alter it and call stream again with the updated Shot

You can also specify a burst = vector of Shots
- e.g., to capture quickly a full HDR stack, or for HDR viewfinder
Sensor produces **Frames**

- **Sensor::getFrame()** is the only blocking call

- A Frame contains
  - image data and statistics
  - the precise time the exposure began and ended
  - the actual and requested (Shot) parameters
  - Tags from Devices (in Frame::tags() dictionary)

- Exactly one Frame for each Shot
  - If image data is lost or corrupted
    - a Frame is still returned
    - with image marked as invalid
    - statistics may be valid
Devices

- **Lens**
  - **focus**
    - measured in diopters: $d \times f = 1\text{m}$
      - $20\text{D} => f = 5\text{cm}, \quad 0\text{D} => f = \infty$
    - the lens starts moving (at specified speed) in the background
  - **focal length (zooming factor)** (fixed on N900)
  - **aperture** (fixed on N900)

- **Flash**
  - fire with a specified brightness and duration

- **Other Devices can be created**
  - FCam example 6 creates a Device for playing the click sound
**Actions allow Devices to coordinate**

- Devices may have a set of Actions, with:
  - start time w.r.t. image exposure start
  - `Action::doAction()` to initiate the action
  - a latency field
    - indicates the delay between the method call and the action begin

- **Shots perform Actions during the exposure**
  - with predictable latency Actions can be precisely scheduled
    - e.g., the timing of Flash in second-curtain sync must be accurate to within a millisecond
Frames are tagged with metadata after they leave the pipeline.

Devices need to keep a short state history that matches with time stamps.

Lens and Flash tag each Frame with their state, making writing an autofocus algorithm straightforward because the focus position of the Lens is known for each Frame.

Other appropriate uses of Tags include sensor fusion.
Tegra implementation of FCam

- NVIDIA Tegra 3
- ISP
- NVIDIA kernel driver
- NVIDIA camera driver
- OpenMax II
- Handler frame assembly
- In-flight shadow queue
- Frame queue
- Setter configuration, timing
- Action device triggering
- Additional frame taps
- Frames to application
- Shots from application

Legend:
- Camera hardware
- SoC component
- Kernel object
- Driver callback
- Runtime thread
- FCam API object

Image data flow
Metadata flow
Image and metadata flow
Control flow
FCam image capture on Tegra (simplified)

1. Request comes in from client
FCam image capture on Tegra (simplified)

1. Request comes in from client
2. Request is put into request queue
1. Request comes in from client
2. Request is put into request queue
3. Setter reads request from queue
FCam image capture on Tegra (simplified)

1. Request comes in from client
2. Request is put into request queue
3. Setter reads request from queue
4. Setter computes timing for possible actions and puts actions in queue
FCam image capture on Tegra (simplified)

1. Request comes in from client
2. Request is put into request queue
3. Setter reads request from queue
4. Setter computes timing for possible actions and puts actions in queue
5. Setter computes ETA for the image data from ISP and puts request info into in-flight shadow queue
1. Request comes in from client
2. Request is put into request queue
3. Setter reads request from queue
4. Setter computes timing for possible actions and puts actions in queue
5. Setter computes ETA for the image data from ISP and puts request info into in-flight shadow queue
6. Setter sets the sensor parameters according to the request
FCam image capture on Tegra (simplified)

1. Request comes in from client
2. Request is put into request queue
3. Setter reads request from queue
4. Setter computes timing for possible actions and puts actions in queue
5. Setter computes ETA for the image data from ISP and puts request info into in-flight shadow queue
6. Setter sets the sensor parameters according to the request
7. Actions are triggered from the action queue at correct time by the Action thread and handled by Devices
FCam image capture on Tegra (simplified)

1. Request comes in from client
2. Request is put into request queue
3. Setter reads request from queue
4. Setter computes timing for possible actions and puts actions in queue
5. Setter computes ETA for the image data from ISP and puts request info into in-flight shadow queue
6. Setter sets the sensor parameters according to the request
7. Actions are triggered from the action queue at correct time by the Action thread and handled by Devices
8. Handler thread reads incoming image data and metadata, connects them with the corresponding request in in-flight queue, and gets Tags from Devices
FCam image capture on Tegra (simplified)

1. Request comes in from client
2. Request is put into request queue
3. Setter reads request from queue
4. Setter computes timing for possible actions and puts actions in queue
5. Setter computes ETA for the image data from ISP and puts request info into in-flight shadow queue
6. Setter sets the sensor parameters according to the request
7. Actions are triggered from the action queue at correct time by the Action thread and handled by Devices
8. Handler thread reads incoming image data and metadata, connects them with the corresponding request in in-flight queue, and gets Tags from Devices
9. Handler puts the assembled Frame object into Frame queue for client
FCam image capture on Tegra (simplified)

1. Request comes in from client
2. Request is put into request queue
3. Setter reads request from queue
4. Setter computes timing for possible actions and puts actions in queue
5. Setter computes ETA for the image data from ISP and puts request info into in-flight shadow queue
6. Setter sets the sensor parameters according to the request
7. Actions are triggered from the action queue at correct time by the Action thread and handled by Devices
8. Handler thread reads incoming image data and metadata, connects them with the corresponding request in in-flight queue, and gets Tags from Devices
9. Handler puts the assembled Frame object into Frame queue for client
Android Camera

Camera HAL Implementation

Application Framework

Request Queue

Camera Apps

Requests

Frame Metadata Queue

Image Buffers

Metadata

Camera2 API Core Operation Model

1 Request → 1 image captured →
1 Result metadata + N image buffers

CaptureRequest

Per-frame settings

Target Surface

Target Surface

CameraDevice

capture()

Pending request queue

Request

Request

Request

Request

Configured output surfaces

Surface

Surface

Surface

In-flight capture queue

Output image queues

Request

Request

Request

Request

Original CaptureRequest

CaptureResult

Settings used by hardware

Hardware status

Camera hardware

onCaptureComplete()