The FCam API for Programmable Cameras

Sung Hee Park
Stanford University
shpark7@stanford.edu

Andrew Adams
Stanford University
abadams@stanford.edu

Eino-Ville Talvala
Stanford University, now at Google Inc.
etalvala@google.com

ABSTRACT

The FCam API is an open-source camera control library, enabling precise control over a camera's imaging pipeline. Intended for researchers and students in the field of computational photography, it allows easy implementation of novel algorithms and applications. Currently implemented on the Nokia N900 smartphone, and a custom-built "Frankencamera", it has been used in teaching at universities around the world, and is freely available for download for the N900.

This paper describes the architecture underlying the API, the design of the API itself, several applications built on top of it, and some examples of its use in education.

Categories and Subject Descriptors

I.4.1 [**Image Processing and Computer Vision**]: Digitization and Image Capture—*Digital Cameras*

General Terms

Experimentation

1. INTRODUCTION

The transition from film to digital cameras that has taken place over the last decade has opened new possibilities for photography. By incorporating computation into the process of creating a photograph we can extend the dynamic range, alter the depth of field, or reduce noise in dark environments. This field has come to be called computational photography. Many of these algorithms operate by acquiring a burst of images with precisely varying camera settings, and then fusing that burst into a single output photograph better than any of the inputs. One key requirement for high-quality results is that the burst must be captured in as short a time as possible.

However, the field has had limited impact to date, as current cameras are not programmable at all, and camera-equipped smartphones use APIs that are incapable of cleanly expressing these algorithms. Typical mobile imaging APIs do not give the programmer control over fundamental sensor parameters like exposure time, and even worse, act adversarily by running uncontrollable metering and focus algorithms.

These APIs typically operate the camera in either a streaming mode suitable for viewfinding, in which changes to camera parameters take effect an unknown number of frames in the future, or in a still capture mode which cannot quickly capture bursts of frames with varying settings. Neither mode is useful for much of computational photography.

To address this need, we developed the FCam API for programmable cameras. The goal for the API is to provide intuitive camera programming model that enables programmers to easily prototype new ideas for computational photography applications. The API makes it easy to precisely control all parameters of the camera, and facilitates capturing bursts of images with deterministically-varying settings at full sensor frame rate.

Such control enables many new applications such as HDR viewfinding, advanced low-light imaging, or synthetic aperture photography, in a straightforward way. The API currently runs on two platforms: The Nokia N900 smartphone, and the custom-built "Frankencamera" [2, 4]. It is written for C++, and was designed at Stanford University in collaboration with Nokia Research Center Palo Alto.

In this paper, we describe the software architecture behind the API, demonstrate use of the API with some code examples and sample applications, and describe the API's use in education. We have found FCam particularly useful in graduate-level classes teaching computer vision and computational photography, and it has been widely used for this purpose over the last year.

2. ARCHITECTURE

The Frankencamera architecture departs from the typical model of the camera sensor and imaging hardware exposed by mobile imaging APIs. Instead of viewing the imaging pipeline as a free-running source of image data, the Frankencamera architecture sees it as a pipeline that transforms requests into images. For each request fed into the pipeline, one output image is produced, in order. The stages of the imaging pipeline themselves store no state — everything needed to specify the image capture and post-processing parameters for a given frame is bundled into the requests, including a list of actions that need to be performed concurrently to sensor exposure by other parts of the camera hardware (e.g. firing a flash). Figure 1 illustrates the overall design.

MM'11, November 28–December 1, 2011, Scottsdale, Arizona, USA.

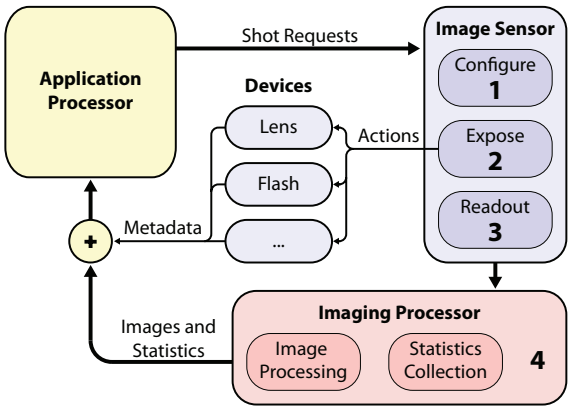


Figure 1: The Frankencamera architecture treats the image sensor as a pipeline, which converts requests for images into images and metadata on a one-for-one basis.

The Sensor.

The sensor simply converts requests into raw image data, matching the requested parameters as closely as possible. The images produced are bundled together with the actual parameters used into a "frame" that is passed down the imaging pipeline. The sensor also triggers device-specific actions at appropriate times during image capture.

Fixed-Function Imaging Processor.

Most cameras include dedicated fixed-function hardware for processing raw camera data efficiently. In the FCam architecture, such imaging processors can transform the raw image data into a form more suitable for display or later processing, but the raw data must always be available to the application if requested. In addition, the imaging processor produces various useful image statistics such as intensity histograms or sharpness maps. These are included in the final frame sent to the application to assist with control algorithms like metering and autofocus.

Other Devices.

A typical camera has many different pieces of hardware that must act in concert to capture a photograph, such as the lens and the flash. In addition, research systems may include novel hardware that must be synchronized with an exposure. These devices advertise a set of "actions" they can perform. Actions can be included in requests, and specify the time that they should take place relative to the start of the exposure.

Devices also produce additional metadata about their state during image capture (such as lens position, or whether the flash fired). This metadata is bundled into the frame sent to the application.

Control Algorithms.

All control algorithms run in the main application processor; there are no hidden daemons or hardware routines controlling metering or autofocus. This gives the camera applications full visibility into the imaging pipeline, and makes it simple for an application to write its own control algorithms. If standard control algorithms are desired, these are available as utility functions.

3. THE FCAM API

In this section we demonstrate use of the API with three brief examples. More examples and complete documentation can be found in [5], and implementation details and design rationale are covered in [4].

HDR Burst.

Our first example rapidly acquires three images at three different exposure times. This is the raw data necessary to make a high-dynamic-range photograph. First we allocate objects representing the image sensor, the three requests (called *Shots* in the API), and the three frames we expect to receive in return.

```
Sensor sensor;
Shot shortReq, midReq, longReq;
Frame short, mid, long;
```

Then we configure each shot to describe the all parameters necessary for its capture. In this example we merely configure exposure time, and allocate some storage for the returned image data. Other available parameters include gain, frame time, and white balance.

```
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);
```

We then *capture* the three shots. This is a non-blocking call which enqueues the requests at the top of the pipeline in Figure 1.

```
sensor.capture(shortReq);
sensor.capture(midReq);
sensor.capture(longReq);
```

Finally, we call *getFrame* to retrieve the resulting images and metadata. This call blocks until the frames are produced. The API guarantees that we will receive one frame per shot requested, in the same order.

```
short = sensor.getFrame();
mid = sensor.getFrame();
long = sensor.getFrame();
```

HDR Viewfinding.

The above code captures a high-dynamic-range burst of images. We will now build this into a full HDR camera application by adding a high-dynamic-range viewfinder. Our viewfinding algorithm will instruct the sensor to alternate between two exposure times and fuse each returned pair of images for display (as in [3]). To accomplish this case we first create a vector of two shots with our two exposure times:

```
vector<Shot> hdr(2);
hdr[0].exposure = 40000;
hdr[1].exposure = 10000;
```

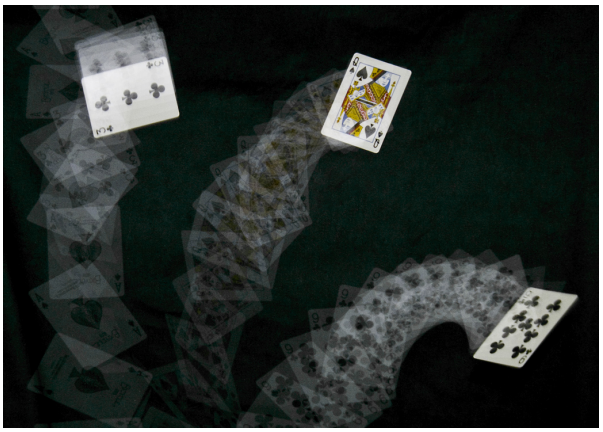


Figure 2: To capture this image, we attached two flash units to a Frankencamera. We use *actions* to direct one to strobe throughout the exposure, and the other to fire brightly once at the end of the exposure.

We then *stream* this pair of shots. This produces an endless stream of frames which alternates between the two shots in the vector. We *stream* instead of *capture* in order to keep the imaging pipeline full and run at video rate without dropping frames. This kind of alternating stream is almost impossible to achieve with conventional camera APIs.

```
sensor.stream(hdr);
```

We then enter a loop, retrieving each pair of frames in the same order as requested.

```
while(1) {
  Frame longExp = sensor.getFrame();
  Frame shortExp = sensor.getFrame();
```

Our high-dynamic range viewfinder will need to perform metering. The short exposure should meter to capture the highlights, and the long exposure should meter to capture the shadows. Fixed-function imaging processors compute histograms which we attach to each frame, so we need not inspect the image data itself in order to meter. We can compute a new exposure for each frame from the histogram and current exposure time alone. The *stream* call makes a copy of the shots passed to it, so after updating the exposure times we must call *stream* again to effect our change. After metering we fuse the two images for display and continue our loop.

```
  hdr[0].exposure = meterLong(longExp.histogram(),
                              longExp.exposure());
  hdr[1].exposure = meterShort(shortExp.histogram(),
                               shortExp.exposure());
  sensor.stream(hdr);
  overlayWidget.display( fuse(longExp, shortExp) );
}
```

Second-Curtain Sync Flash.

Our final example demonstrates the use of actions. This example fires a flash at the end of the exposure (this is known as second-curtain sync). First we create and configure our shot and the object that represents the flash.

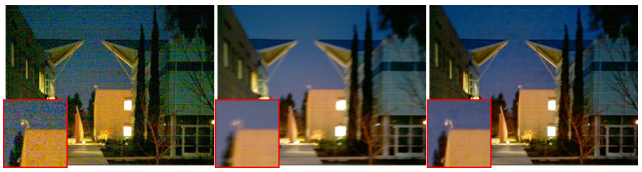


Figure 3: Low-light imaging. The FCam API is used to create a low-light camera mode. A short-exposure noisy image (left), and a long-exposure blurry image (center) are captured, and combined into a high-quality result (right) directly on the camera.

```
Shot flashShot;
flashShot.exposure = 100000; // 0.1 seconds
Flash flash;
```

We then create and configure the *action* that represents firing the flash. We set it to trigger at the end of the exposure. On the N900 implementation, FCam can trigger actions with an accuracy of ±30 microseconds, which is sufficient for photography.

```
Flash::FireAction fire(&flash);
fire.duration = 1000; // 1 ms
fire.brightness = flash.maxBrightness();
fire.time = flashShot.exposure - fire.duration;
```

Finally, we attach this action to the shot, and tell the sensor to *capture* this shot as in the previous examples. Using this kind of program, we produced the image in Figure 2.

```
flashShot.addActions(fire);
sensor.capture(flashShot);
Frame flashFrame = sensor.getFrame();
```

4. SAMPLE APPLICATIONS

To test the FCam API, we developed several sample applications. These were meant to demonstrate features of the API, and be representative of recent work in computational photography. Here, we'll discuss two examples.

Low-noise Viewfinding and Capture.

Taking high-quality photographs in low light is a challenging task. To achieve the desired image brightness, one must either increase gain, which increases noise, or increase exposure time, which introduces motion blur and lowers the frame rate of the viewfinder. In this application, the FCam API is used to implement a low-light camera mode, which augments viewfinding and image capture using the algorithms of Adams et al. [1] and Tico and Pulli [6], respectively. In brief, the application takes a burst of two images — one noisy, with short exposure time and high gain, and a second blurry image, with a long exposure time. These images are then combined into a high-quality result on the device.

Lucky Imaging.

Long-exposure photos taken without use of a tripod are usually blurry, due to natural hand shake. However, hand shake varies over time, and a photographer can get "lucky" and record a sharp photo if the exposure occurs during a period of stillness (Figure 4). Our lucky imaging application

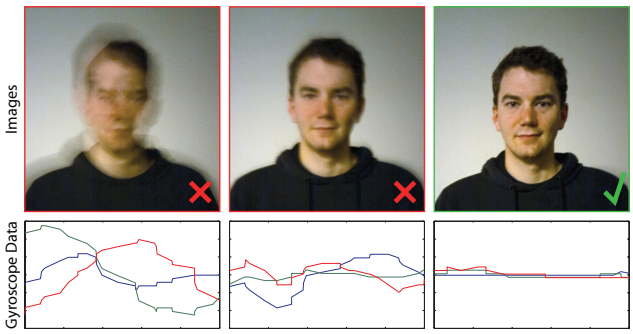


Figure 4: Lucky imaging. An image stream and 3-axis gyroscope data for a burst of three images with half-second exposure times. The Frankencamera API makes it easy to tag image frames with the corresponding gyroscope data.

uses an experimental 3-axis gyroscope affixed to the front of the N900 to detect hand shake. Newer smartphones, such as the Apple iPhone 4, have built-in gyroscopes of similar quality. By treating the gyroscope as a *device* in our API, we can easily attach a synchronized gyroscope trace to each returned frame. We then stream full-resolution images until we receive a frame with a trace indicating that there was little motion during that exposure.

5. EDUCATIONAL USE

With the assistance of Nokia, sets of Nokia N900s along with the FCam API have been distributed to universities around the world, to be used in teaching. Universities that have received devices include Brown University; Cornell; Carnegie Mellon; École Polytechnique Fédérale de Lausanne; ETH Zürich; Fachhochschule Brandenburg; Georgia Tech; IMPA (Brazil); Johannes Kepler University; National University of Ireland; MIT; Oregon State University; Rochester Institute of Technology; University of California, Santa Barbara; University of California, Santa Cruz; University of Catania; University of Hong Kong; University of New Mexico; University of Virginia; University of Zaragoza; and Stanford University.

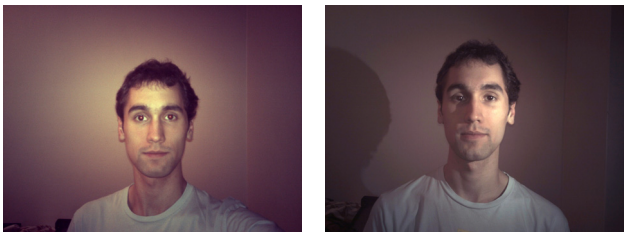
As an example, at Stanford University FCam was used in a graduate-level course on computational photography. As a first assignment to learn the API, students were asked to write an auto-focus algorithm for the N900 in one week. The results were encouraging; the top students' methods were faster and more robust than the N900's built-in auto-focus routine.

After the initial assignment, the students had one month to complete a final project. Two of the resulting applications are shown in Figures 5 and 6.

6. CONCLUSION

FCam is an open-source API intended for researchers, developers, educators, and hobbyists interested in programming their camera. Since its release in mid-2010, the FCam source package has been downloaded over 7,000 times, and the compiled library has been downloaded over 100,000 times. The library is available under the BSD license.

In the future we intend to expand the reach of the API by porting it to other mobile platforms. Just as importantly, we hope the API demonstrates to camera and smartphone manufacturers the benefits of opening up the camera subsystem to third-party developers and the open-source community at large.



(a) On-camera flash. (b) Flash from helper N900.

Figure 5: Remote flash. In this project, the students synchronized two N900s so that the other smartphone fired its flash when the first took a picture. *Images courtesy of D. Keeler and M. Barrientos, with post-processing for color balance and contrast.*

(a) A single captured frame. (b) 16 frames merged.

Figure 6: Painted aperture. By capturing a set of images from slightly different viewpoints, an image that appears to come from a camera with a much shallower depth-of-field can be created. *Images courtesy of E. Luong.*

7. REFERENCES

[1] A. Adams, N. Gelfand, and K. Pulli. Viewfinder Alignment. *Computer Graphics Forum (Eurographics 2008)*, 27(2):597–606, 2008.

[2] A. Adams, E.-V. Talvala, S. H. Park, D. E. Jacobs, B. Ajdin, N. Gelfand, J. Dolson, D. Vaquero, J. Baek, M. Tico, H. P. A. Lensch, W. Matusik, K. Pulli, M. Horowitz, and M. Levoy. The Frankencamera: An experimental platform for computational photography. *ACM Transactions on Graphics (SIGGRAPH 2010)*, 29(4):29:1–29:12, 2010.

[3] N. Gelfand, A. Adams, S. H. Park, and K. Pulli. Multi-exposure imaging on mobile devices. In *Proceedings of the ACM International Conference on Multimedia*, pages 823–826, 2010.

[4] E.-V. Talvala. *The Frankencamera: Building a programmable camera for computational photography.* PhD thesis, Stanford University, 2011.

[5] E.-V. Talvala and A. Adams. The FCam API. http://fcam.garage.maemo.org, 2010.

[6] M. Tico and K. Pulli. Image enhancement method via blur and noisy image fusion. In *Proceedings of the 16th IEEE International Conference on Image Processing (ICIP)*, pages 1521–1524, 2009.