The Frankencamera:

An Experimental Platform for Computational Photography

Andrew Adams  Eino-Ville Tavala  Sung Hee Park
David E. Jacobs  Boris Ajdin  Natasha Gelfand
Jennifer Dolson  Daniel Vaquero  Jongmin Baek
Marius Tico  Hendrik P. A. L.  Wojciech Matusik
Kari Pulli  Mark Horowitz  Marc Levoy

Young Bin Kim

2010/09/29
Korea University
Computer Graphics Lab.

SIGGRAPH 2010
Computational Photography
Computational Photography

- **KCGS 2010 Workshop**
  - 영상기기(영상, 비디오) + 컴퓨터(계산, 처리) = Computational Photography
  - 영상/비디오 복원 및 개선
    - 영상 블러 제거
    - 렌즈 왜곡 보정
    - HDR 영상 복원
    - ...
  - 새로운 카메라 기술을 이용한 영상/비디오 획득
    - 비디오 합성
    - 고품질 영상/비디오 기반 컨텐츠
    - ...

KUCG | Korea University Computer Graphics Lab.

Young Bin Kim | 2010/09/29 | # 3
Abstract

• **Problems of Computational Photography**
  - Lack of a portable, programmable camera
    - Sufficient image quality
    - Computing power

• **The Frankencamera**
  - Open architecture and API
    - Based on Linux
    - API for C++
    - Incorporate and synchronize external hardware
    - ...

---
Introduction
Introduction

• **Cameras are closed platforms**
  - Hard to incrementally deploy these techniques
  - Researchers to test them in the field
    • No strong incentive to address their artifacts

• **Other platforms (ex – Smartphones)**
  - (+) Encourage applet creation
  - (+) Increasingly capable imaging hardware
  - (-) Programming interface to the imaging system is highly simplified
  - (-) Mimicking the physical interface of a point-and-shoot camera
Introduction

• Camera architecture and API
  ▪ Flexible enough
    • Implement most of the techniques
  ▪ Precise enough
    • Implementations can be built and verified for it
  ▪ Easy to program
Introduction

• **Content**
  2. Prior Work
     • Previous work in this area
  3. The Frankencamera Architecture
     • Camera architecture in more detail
       = F2 and Nokia N900
  4. Programming the Frankencamera
     • How to program for architecture
  5. Applications
     • Six computational photography applications
  6. Conclusion
Prior Work
Prior Work

• Prior work to camera platforms
  ▪ Consumer cameras
  ▪ Smartphones
  ▪ Smart cameras
Prior Work

• **Consumer cameras**
  - Steadily expanding the range of features
    • Ex) Casio EX-F1
      - Can capture bursts of images at 60 fps at a 6-megapixel resolution
      - Can be computationally combined into a new image directly
        • The camera software cannot be modified
        • No Additional features can be explored
Prior Work

- **Consumer cameras**
  - DSLR and point-and-shoot cameras
    - Vendor-supplied firmware
    - Released SDKs
      - Allow to control using an external PC
      - Can be useful for some computational photography applications
      - No access to lower layers
        - Metering
        - Auto-focus algorithms
        - etc.
Prior Work

• **Consumer cameras**
  - Reverse-engineered the firmware
    - Canon Hack Development Kit
      - Non-destructively replaces the original firmware
      - Can adding features
        - custom burst modes, motion-triggered photography, time-lapse photography, etc.
    - Magic Lantern project
      - enhanced firmware for Canon 5D Mark II DSLRs
    - Remove the need to attach a PC and the latency problem
      - Same level of control as the official SDK
      - Lower levels of the camera are still a black box
Prior Work

- **Smartphones**
  - Programmable cell phones
    - Encourage third-party applications
  - Capable of capturing still photographs and videos
    - Quality comparable to point-and-shoot cameras
  - Contain numerous input and output devices
    - Touch screen, audio, buttons, GPS, compass, accelerometers, etc.
  - compact and portable
  - provide limited interfaces to their camera subsystems
Prior Work

- **Smartphones**
  - Apple iPhone 3Gs, Google Nexus One, Nokia N95
  - Variable-focus lenses
  - Megapixel image sensors
  - None allow control over absolute exposure time
  - None allow retrieval of raw sensor data
  - Narrow range of computational photography
Prior Work

- **Smart cameras**
  - **Smart cameras**
    - Image sensors combined local processing, storage, or networking
    - Embedded computer vision systems
      - Smart cameras as embedded systems
        - [Wolf, W., Ozer, B., Lv, T. 2002., Computer 35]
      - Distributed embedded smart cameras for surveillance applications
Prior Work

- **Smart cameras**
  - Smart cameras
    - Complete control over the imaging system
    - Implementing frame capture
    - Low-level image processing
    - Vision algorithms
      - background subtraction, object detection, object recognition, etc.
Prior Work

- **Smart cameras**
  - Research systems
    - Cyclops, MeshEye, Philips wireless smart camera motes
  - Commercial systems
    - National Instruments 17XX, Sony XCI-100, Basler eXcite
  - CMUcam3
    - Open-source embedded vision platform
  - Elphel 353
    - Network cameras
  - Frankencamera prototype
    - Built around an Elphel 353
    - Streamed image data from the Elphel 353 over Ethernet
    - Nokia N800 served as the viewfinder and UI
Prior Work

• **Smart cameras**
  - **Main limitation**
    - Not complete cameras
      - Few support synchronization with other I/O devices
      - None contain a viewfinder or shutter button
    - Network latency problem
Prior Work

- Frankencamera platforms
  - Attempt to provide everything
    - Full access to the imaging system
      - Like a smart camera
    - A full user interface (Viewfinder and I/O interfaces)
      - Like smartphone
    - The ability to be taken outdoors, untethered
      - Like a consumer camera
The Frankencamera Architecture
The Frankencamera Architecture

• A programmable Camera Platform
  1. Should be open, all the way down
  2. Should be able to capture or stream bursts of images with varying settings at full frame rate
  3. Should have enough compute and memory
  4. Should be easy to program
  5. Should be a credible walking around camera
The Frankencamera Architecture

• The Frankencamera Abstract Architecture

![Diagram of the Frankencamera Architecture](image_url)
The Frankencamera Architecture

- **The Image Sensor**
  - Converts *requests* for images into images
  - A *request* for an image specifies all parameters
The Frankencamera Architecture

- The Image Signal Processor (ISP)
  - Receives sensor data, and optionally transforms it
  - Produce raw data and demosaicked format
  - Create histograms, other statistics for each image
The Frankencamera Architecture

- **Devices**
  - Schedule **Actions** to be triggered at a given time
  - Can **tag** returned images with extra metadata

![Diagram of Frankencamera Architecture]
The Frankencamera Architecture

• Discussion

  ▪ Pipelined architecture is simple
  ▪ Programmer has full control over sensor settings
  ▪ Full access to the supplemental statistics the ISP computes for each frame
The Frankencamera Architecture

- The F2
The Frankencamera Architecture

• The F2
  ▪ Limitation
    • The sensor size
      = F2 uses a cell-phone-quality image sensor
      = Currently engineering a DSLR quality replacement

  • Resolution switching is slow
    = Due to underlying ISP driver
    = Roughly 700 ms ‘shutter lag’
    = Not fundamental to the architecture, but hard to fix
The Frankencamera Architecture

- The Nokia N900
The Frankencamera Architecture

• The F2 and the Nokia N900
  ▪ Roughly 80MB of free memory
    • Used purely as image buffer
      ___ Represents eight 5-MP images, or 130 VGA frames
  ▪ Data can be written at 20 MB/sec
Programming the Frankencamera
Programming the Frankencamera

• Developing for either Frankencamera
  1. Writes standard C++ code
  2. Compiles it with a cross-compiler
  3. Copies the resulting binary to the device
  4. Programs can then be run over ssh
     or launched directly on the device’s screen
• Standard debugging tools available
Programming the Frankencamera

- “FCam” API
  - Shots
  - Sensors
  - Frames
  - Devices
Programming the Frankencamera

- Shots
  - Example code

```c
Shot shot;
shot.gain = 1.0;
shot.exposure = 10000;
shot.frameTime = 33333;
shot.image = Image(640, 480, UYVY);
shot.histogram.regions = 1;
shot.histogram.region[0] = Rect(0, 0, 640, 480);
```

- Sensor parameters
- Frame rate
- Output resolution format
- Region of histograms
Programming the Frankencamera

• Sensors
  - *capture* a configured shot

```java
Sensor sensor;
sensor.capture(shot);
```

1. When shot reaches the head of the request queue
2. And the sensor is ready to begin configuring
3. Then shot is issued into the pipeline
Programming the Frankencamera

• **Sensors**
  - Problem
    1. The `sensor is ready`
    2. The `request queue is empty`
    3. Then bubble necessarily enters the pipeline

  - *stream* a shot
    • `sensor.stream(shot)`
Programming the Frankencamera

• **Sensors**
  - Capture or stream vectors of shot, or bursts

```cpp
std::vector<Shot> burst(2);
burst[0] = shot;
burst[1] = shot;
burst[1].exposure = burst[0].exposure*2;
sensor.stream(burst);
```

1. Makes a burst from two copies of our shot
2. Doubles the exposure of one of them
3. Uses the sensor’s stream method to create frames
Programming the Frankencamera

- **Frames**
  - Identified by the *id* field of their shot
  - Example code

```cpp
while (1) {
    Frame::Ptr frame = sensor.getFrame();
    if (frame.shot().id == burst[1].id) {
        displayImage(frame.image);
    } else if (frame.shot().id == burst[0].id) {
        unsigned newExposure = metering(frame);
        burst[0].exposure = newExposure;
        burst[1].exposure = newExposure*2;
        sensor.stream(burst);
    }
}
```

- Displays the longer exposure of the two frames
- Uses the shorter of the two to perform metering
Programming the Frankencamera

- **Devices**
  - **Object with methods**
    - Define a set of *actions*
    - Define a set of *tags*
  - **The lens**
    - Initiate a change to any of its three parameters
      - focus (measured in diopters)
        - setFocus
      - focal length
        - setZoom
      - aperture
        - setAperture
Programming the Frankencamera

• Devices
  ▪ The lens
    • Example code (1)

    ```
    Lens lens;
    float speed = (lens.getFocus() - lens.farFocus()) / 2;
    lens.setFocus(lens.farFocus(), speed);
    ```
    
    = Moves the lens from its current position to infinity focus

    • Example code (2)

    ```
    Frame::Ptr frame = sensor.getFrame();
    Lens::Tags *tags = frame->tags(&lens);
    cout << "The lens was at: " << tags->focus;
    ```
    
    = Tag each returned frame with the state
Programming the Frankencamera

• Devices
  ▪ The flash

- Has a single method that tells it to fire
- Specified brightness and duration
- Has methods to query bounds on brightness and duration
- Flashes with more capabilities
- Can be implemented as subclasses of the base flash class
Programming the Frankencamera

- **Included Algorithms**
  - Generic metering and autofocus algorithms
    - Helpful and convenient
    - Custom metering and autofocus algorithms possible

- **Metering**
  - Operates on the image histogram
    1. Attempts to maximize overall brightness
       - Minimizing the number of oversaturated pixels
    2. Takes a pointer to a shot and a frame
    3. Modifies the shot with suggested new parameters
Programming the Frankencamera

• Included Algorithms
  ▪ Autofocus
    • It terminates in at most half a second
    • Quite robust
      1. Sweeping the lens from far focus to near focus
      2. Sweep is complete, or sharpness degrades for several frames in a row
      3. The lens is moved to the sharpest position
Programming the Frankencamera

• Included Algorithms
  ▪ Image Processing
    • Free to use any image processing library
    • Provide methods
      - Save raw files to storage
      - Demosaic
      - Gamma correct
      - Store JPEG images
Programming the Frankencamera

- **Implementation**
  - FCam runs entirely on the ARM CPU
    - Using a small collection of user-space threads
    - Using a modified Linux kernel modules
Programming the Frankencamera

• Implementation
  ▪ Setting Sensor Parameters
    • “Setter” thread
      - Sensor parameter updates
      1. Readout of frame n begins
      2. Post-processing are set for frame n+1
      3. Sets all the parameters for frame n+2

• Add controls to specify the time taken by each individual frame
Programming the Frankencamera

• **Implementation**
  - Handling Image Data
    - “Handler” thread
      1. Receives the V4L2 image buffers
         - Produced by the imaging pipeline
         - Consist of timestamped image data
      2. Queries the imaging processor driver for any requested statistics
      3. Copied into the frame’s desired memory target
      4. Completed FCam frame is placed in the frame queue
      5. Retrieved by the application
Programming the Frankencamera

• Implementation
  • Scheduling Device Actions
    • “Action” thread
      = Highest priority level
      = Manages the timing of scheduled actions
      = Problem occurs
        • Sleeps until several hundred microseconds before the trigger time of the next scheduled action
    = Solution
      • Sacrifice a small amount of CPU time
      • Able to schedule actions with an accuracy of within 20 microsecond
Programming the Frankencamera

- Implementation
  - Performance
    - Streaming 640 X 480 frames at 30 frames per second
      - Uses 11% of the CPU time
    - If image data is discarded
      - Usage drops to 5%
    - Displaying, metering, focusing
      - Do not measurably increase CPU usage
Programming the Frankencamera

• **Implementation**
  - **Discussion**
    - API provide intuitive mechanisms
      - Precisely manipulate camera hardware state over time
    - Programmers can use
      - Other image processing library
      - UI toolkit
      - File I/O
      - Etc.
    - Future image sensors
      - Identify every frame they produce the input and output sides
Applications
Applications

- Six computational photography applications
  - Rephotography
  - IMU-based Lucky Imaging
  - Foveal Imaging
  - HDR Viewfinding and Capture
  - Low-Light Viewfinding and Capture
  - Panorama Capture
Applications

- Rephotography
  - Computational re-photography
Applications

• **Rephotography**
  - In the original system
    • Computation and user interactions take place on a laptop
  - In our implementation
    • Handling user interaction more naturally
      - Through the touchscreen LCD of the N900
Applications

- IMU-based Lucky Imaging

- Attach 3-axis gyroscope to the N900
- Usually done in 10 frames
Applications

- **Foveal Imaging**
  - CMOS image sensors are bandwidth-limited devices
  - Full-sensor-resolution: 12 fps
  - Low resolution: 90 fps
    - Produced by downsampling or cropping on the sensor
Applications

• HDR Viewfinding and Capture
  ▪ Modern cameras don’t include a complete “HDR mode”
    • Automatic metering
    • Viewfinding
    • compositing of HDR shot
  ▪ FCam API to implement such an application
Applications

• HDR Viewfinding and Capture
  ▪ Created completely on-camera, ~1 minute processing time
Applications

- HDR Viewfinding and Capture

```c
#include <FCam/N900.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();
```
Applications

• HDR Viewfinding and Capture

```cpp
#include <FCam/N900.h>
...
vector<Shot> hdr(2);
hdr[0].exposure = 40000;
hdr[1].exposure = 10000;
...
while(1) {
    sensor.stream(hdr);
    Frame longExp = sensor.getFrame();
    Frame shortExp = sensor.getFrame();
    hdr[0].exposure = autoExposeLong(longExp.histogram(),
                                    longExp.exposure());
    hdr[1].exposure = autoExposeShort(shortExp.histogram(),
                                      shortExp.exposure());
    overlayWidget.display( blend(longExp, shortExp) )
}
```
Applications

• **Low-Light Viewfinding and Capture**
  - Combines multiple aligned frames in viewfinder mode
  - High resolution capture combined two captures
    - Noisy short exposure
    - Blurry long exposure
Applications

• Panorama Capture
Conclusion
Conclusion

• **Future imaging platforms should support**
  1. Per-frame resolution switching at video-rate
  2. Imaging processors that support streaming data from multiple image sensors at once
  3. A fast path from the imaging pipeline into the GPU
  4. A feature detector and descriptor generator among the statistics collection modules in the imaging processor
  5. Programmable execution stages replace the fixed-function transformation and statistics generation modules in the imaging path
Conclusion

- Current APIs are bad for computational photography
- Camera platforms should be open