Real-time HDR Tonemapping with a Single Camera

~ Chris McClanahan ~

4/26/2011

HDR $\neq$ Tonemapping

High Dynamic Range Imaging

- Effective blending different exposures
- Basic:
  - $hdr=[\log(img1)+\log(img2)+\log(img3)]/3$
- Pixel values normalized [0- 1.0+], instead of fixed [0- 255]

Tonemapping

- Compressing an HDR image back down to [0-255] range
- Many Artful and Realistic methods exist
  - Tone Mapping Operator (TMO)

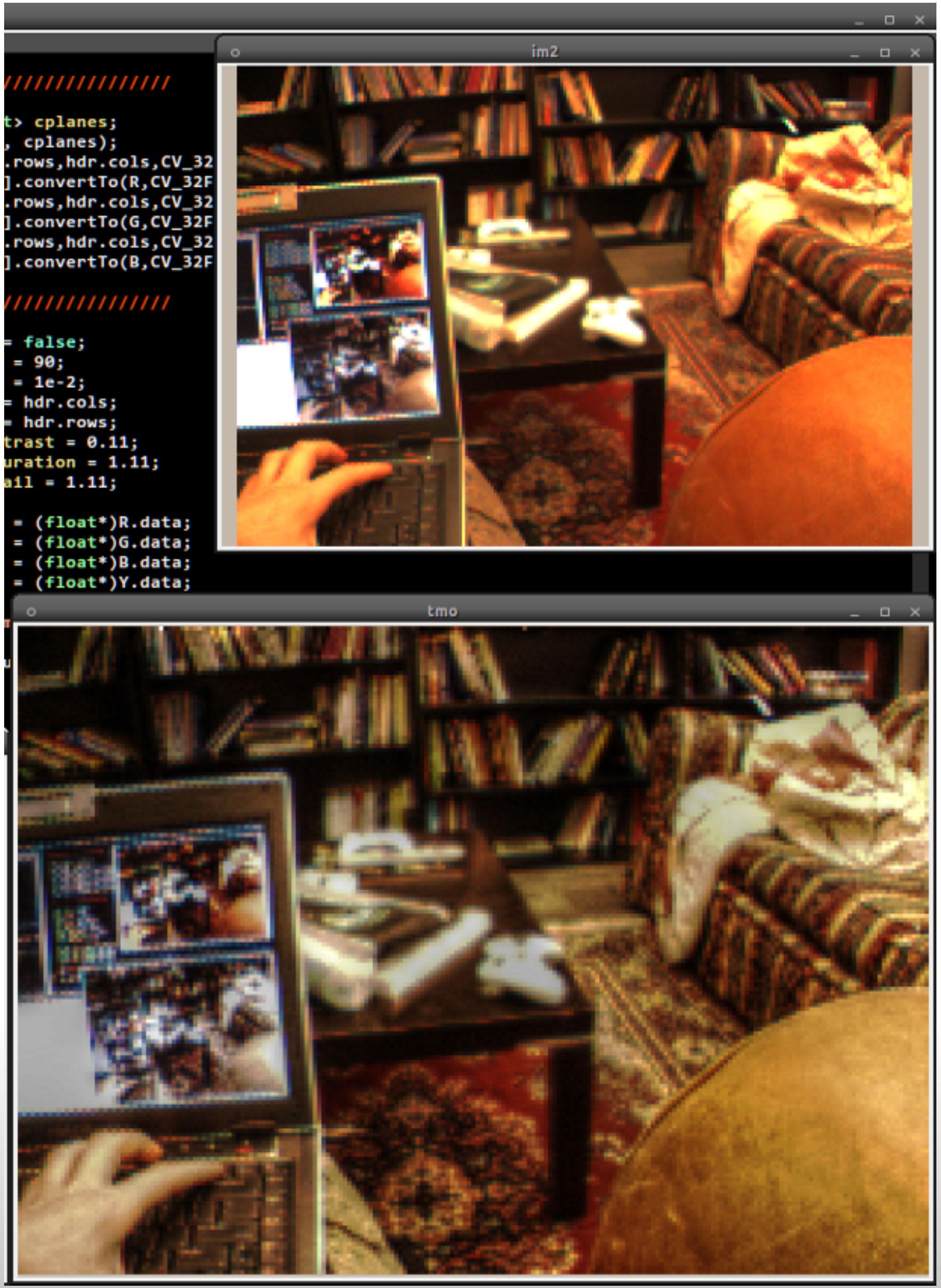
Project Overview

Created an application that uses a single video camera to create tonemapped HDR images in real time.

The app runs on Linux PCs (using USB or Firewire cameras) and Android phones (using the built-in camera).

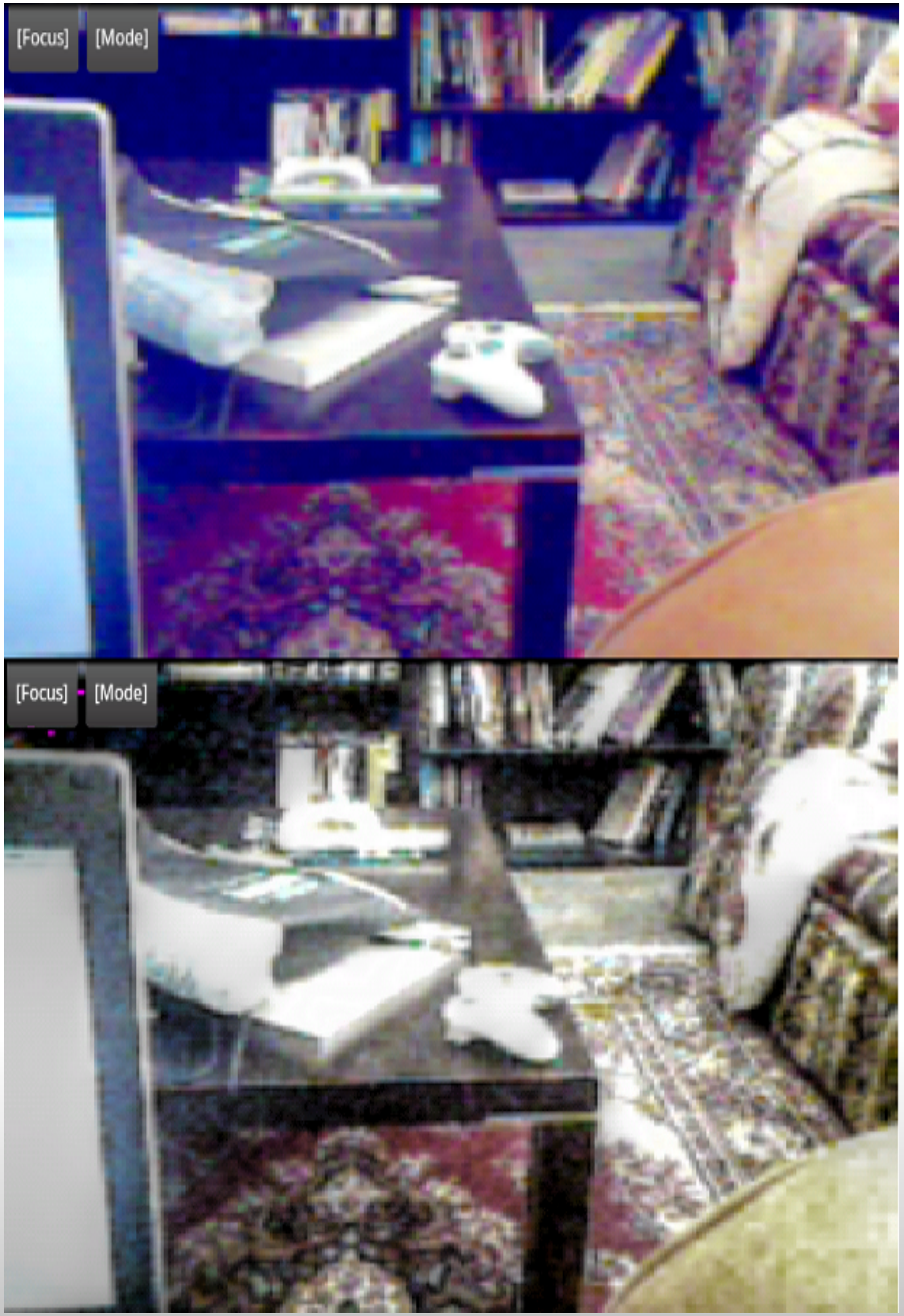
Custom OpenCV code manages the different exposure images, and generates a basic HDR image.

The HDR image is fed to a tonemapping algorithm by Mantiuk et al, creating a ‘ghostly’ effect.



```
////////////////

t> cplanes;
, cplanes);
.rows,hdr.cols,CV_32
].convertTo(R,CV_32F
.rows,hdr.cols,CV_32
].convertTo(G,CV_32F
.rows,hdr.cols,CV_32
].convertTo(B,CV_32F

////////////////

= false;
 = 90;
 = 1e-2;
= hdr.cols;
= hdr.rows;
trast = 0.11;
uration = 1.11;
ail = 1.11;

 = (float*)R.data;
 = (float*)G.data;
 = (float*)B.data;
 = (float*)Y.data;
```

Firewire Camera



Android Camera

Implementation

1. Capture 3 images with varying exposures (low, med, high)
2. hdr=[log(img1)+log(img2)+log(img3)]/3;
3. HDR image fed to a tone mapping algorithm by *Mantiuk et al*

Mantuik et al’s tonemapping operator

- Taken from the Luminance HDR project.
- Two modes:
  - contrast mapping
  - contrast equalization
- 'ghostly' effect
- Computationally intensive
  - Requires severe down-scaling of the raw camera images to keep processing time reasonably fast.

Implementation - Linux

- USB cameras are supported (and captured) by OpenCV
- Firewire cameras handled by custom libdc1394 wrapper
- Most USB cameras only support changing the brightness
  - (not exposure)
  - generates a faux-HDR image that then gets tonemapped.
- An AVT Guppy machine vision firewire camera
  - allows changing the shutter speed and adjusting gain.
  - This camera produced much better results than any webcam tested.

Implementation - Android

- Built-in camera controlled via Java Android camera APIs
- Massive delay between setting the exposure, and when the camera actually gets to that exposure (if it even does).
- An arbitrary number of dummy frames are discarded before grabbing an image
  - attempt to give the camera time to adjust
  - waiting for the camera’s exposure change takes a while
  - exaggerates an already slow image processing loop.

Application Details

Pros / Features:

- Single camera, live HDR
- Mantuik TMOs:
  - Contrast Mapping (faster, but less dramatic)
  - Contrast Equalization (slower, better looking)
- Cross platform
  - Android / Linux
- Various camera support
  - USB / Firewire / Android
- No image alignment pre-processing needed (assuming little camera movement)
- OpenCV + OpenMP

Cons / TODOs:

- Very low resolution
- Low frame rate
  - exposure change time limits frame rate
- Android’s camera exposure change is terribly slow
- No fancy GUI
- Manual adjustment of camera settings required
  - trial-and-error based
- Results extremely dependent on quality of the camera’s exposure changes

Resources

- All the code for this project in my Google Code site:
  - ViewerCV (Android)
  - rttmo (Linux)

Obligatory blog post about this project