Construction, Quality Assessment, and Applications of Pixel Value Error PDF Models

Henry (Hank) Dietz

IPAS-225: 9:30 – 9:50 AM, February 4, 2025

University of Kentucky
Electrical & Computer Engineering

Abstract

Increasingly sophisticated algorithms, including trained artificial intelligence methods, are now widely employed to enhance image quality. Unfortunately, these algorithms often produce somewhat hallucinatory results, showing details that do not correspond to the actual scene content. It is not possible to avoid all hallucination, but by modeling pixel value error, it becomes feasible to recognize when a potential enhancement would generate image content that is statistically inconsistent with the image as captured. An image enhancement algorithm should never give a pixel a value that is outside of the error bounds for the value obtained from the sensor. More precisely, the repaired pixel values should have a high probability of accurately reflecting the true scene content.

The current work investigates computation methods and properties of a class of pixel value error model that empirically maps a probability density function (PDF). The accuracy of maps created by various practical single-shot algorithms is compared to that obtained by analysis of many images captured under controlled circumstances. In addition to applications discussed in earlier work, the use of these PDFs to constrain AI-suggested modifications to an image is explored and evaluated.

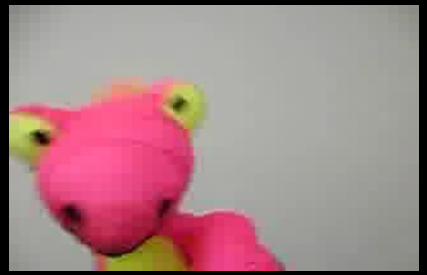
Why Model Pixel Value Error?

- To recognize when scene content has or has not changed in a time series of captures
- To distinguish noise from actual scene content
- To constrain image enhancements (even AI modifications) to be consistent with knowable properties of the scene photographed
- To evaluate quality of images and regions within images

Recognizing When Scene Content Changes

- EI2016: Scene appearance change as framerate approaches infinity
- EI2017: Temporal super-resolution for time domain continuous imaging
- EI2017: TIK: a time domain continuous imaging testbed using conventional still images and video

Original 240FPS, 346º



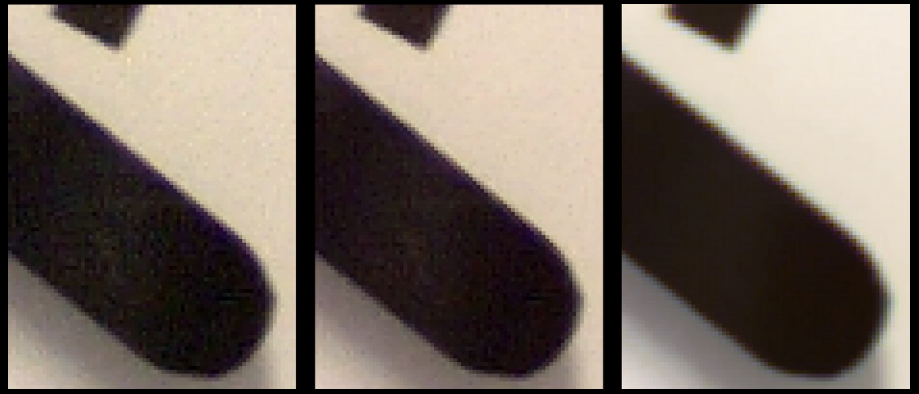
TIK 240FPS, 360º

TIK 29.97FPS, 360º

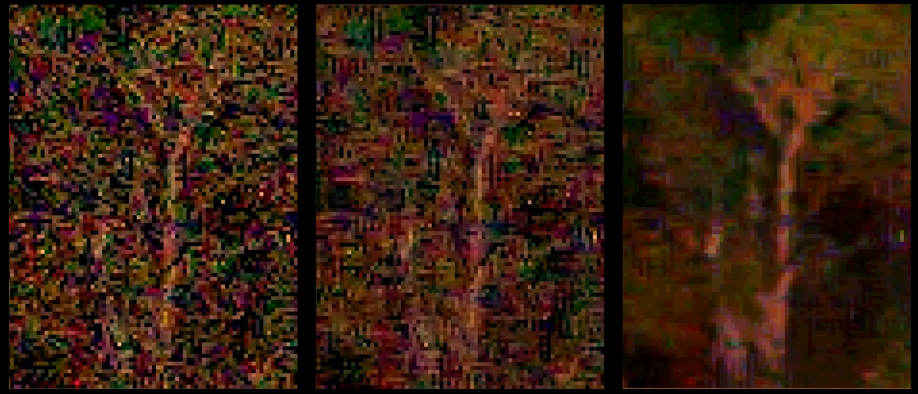
Distinguishing Noise From Scene Content

- EI2017: Refining raw pixel values using a value error model to drive texture synthesis
- EI2022: An improved raw image enhancement algorithm using a statistical model for pixel value error

Raw KREMY ’17 KREMY ’22



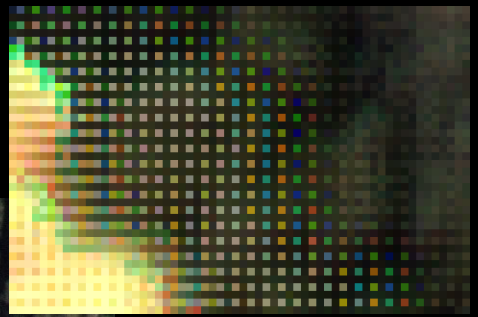
Raw KREMY ’17 KREMY ’22



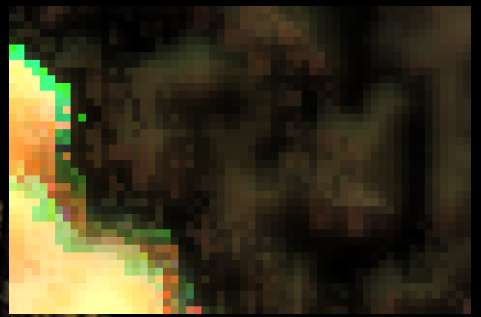
Constraining Enhancements To Be Scene-Consistent

- EI2024: Leveraging pixel values certainty in pixel-shift and other multi-shot super-resolution processing

PixelShift2DNG



Parsek



Pixel Value Error Models

- Noise in pixel values is complex
  - Caused by photon shot noise, sensor, and processing
  - Depends on camera settings and environment
  - Might vary with color channel, pixel position, neighboring values
- **Probability Density Function (PDF) value error model**
  - Approximates probability pixel value v_{ideal} will be read as v_{read}
  - Probability is a 2D function of (v_{ideal}, v_{read}) – visualized as an image with probability as brightness
  - Red, Green, and Blue PDFs visualized as one RGB image

How To Make A Pixel Value Error Model

- Empirically determined; too complex for analytic solution
- Can measure v_{read}, but how can we determine v_{ideal}
  - Use a calibrated scene – not practical
  - Average over many samples *should* converge on v_{ideal}
  - Don’t – measure *variance* without knowing v_{ideal}
- We have tried many different algorithms over the years, but never tested them side-by-side until now

How `errpdf` Makes A Pixel Value Error Model

- The `errpdf` program:
  - Mostly **C++** code using **OpenMP** and `OpenCV` library
  - Input is a time sequence of images of a static scene
  - Processes **JPEG** images directly
  - Uses `dcraw` or `unprocessed_raw` to help read raw images
  - Processing can be CFA pattern aware even for **JPEG**s
- Averages all images to create reference image of v_{ideal} values
- Outputs PDF model as RGB **PNG** image for each algorithm

3x4x4 = 48 Algorithms Implemented In `errpdf`

- Basic approaches used for **JPEG**, **JPEG** with CFA, and raw:
  - **`ref`**: records pixel changes WRT v_{ideal} computed by averaging
  - **`pair`**: differences image pairs as v_0, v_1 without knowing v_{ideal}
  - **`one`**: differences most similar neighbor pixel within one image
  - **`max`**: differences all neighbor pixels with similar neighbors
- Processing applied to histogram to encode PDF:
  - Normalize probabilities so that v_{ideal} value is 100%
  - **`mono`**: make values monotonically approach v_{ideal}, normalize
  - **`box`**: (v_0, v_1) becomes box covering (v_0, v_1) & (v_1, v_0), normalize
  - **`smooth`**: iteratively smooth along diagonal, normalize

Compute Cost Of Algorithms Implemented In `errpdf`

- Basic approaches used for **JPEG**, **JPEG** with CFA, and raw:
  - **`ref`**: records pixel changes WRT v_{ideal} computed by averaging
  - **`pair`**: differences image pairs as v_0, v_1 without knowing v_{ideal}
  - **`one`**: differences most similar neighbor pixel within one image
  - **`max`**: differences all neighbor pixels with similar neighbors
- Processing applied to histogram to encode PDF:
  - Normalize probabilities so that v_{ideal} value is 100%
  - **`mono`**: make values monotonically approach v_{ideal}, normalize
  - **`box`**: (v_0, v_1) becomes box covering (v_0, v_1) & (v_1, v_0), normalize
  - **`smooth`**: iteratively smooth along diagonal, normalize

PDF Quality From Algorithms Implemented In `errpdf`

- Basic approaches used for **JPEG**, **JPEG** with CFA, and raw:
  - **`ref`**: records pixel changes WRT v_{ideal} computed by averaging
  - **`pair`**: differences image pairs as v_0, v_1 without knowing v_{ideal}
  - **`one`**: differences most similar neighbor pixel within one image
  - **`max`**: differences all neighbor pixels with similar neighbors
- Processing applied to histogram to encode PDF:
  - Normalize probabilities so that v_{ideal} value is 100%
  - **`mono`**: make values monotonically approach v_{ideal}, normalize
  - **`box`**: (v_0, v_1) becomes box covering (v_0, v_1) & (v_1, v_0), normalize
  - **`smooth`**: iteratively smooth along diagonal, normalize

Example: “Art” Scene raw Sony A7RV @ 12800

A7RV raw @ 12800

`ref smooth`

`pair smooth`

`one smooth`

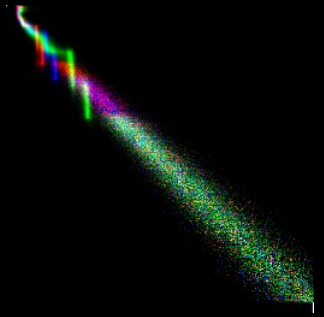
`max smooth`

- `ref smooth`: accept this as ground truth
- `pair smooth`: tends to overestimate noise
- `one smooth`: tends to underestimate, especially using JPEG
- `max smooth`: tends to overestimate noise

Example: “Art” Scene raw Sony A7RV @ 12800

A7RV raw @ 12800

`ref smooth`



`ref`

`ref mono`

`ref box`

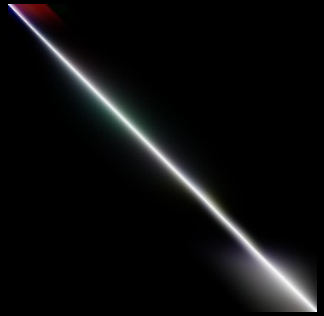
- **`ref smooth`**: accept this as ground truth
- **`ref`**: PDF is noisy depending on scene content
- **`ref mono`**: tends to give too much weight to outliers
- **`ref box`**: slightly splotchy depending on scene content

Sony A7RV and Canon SX530 raw, JPEG, & “Card”

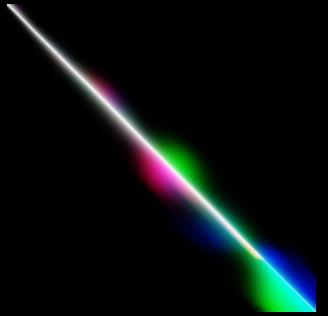
A7RV @ 100

SX530 @ 100

SX530 @ 1600



... from JPEG



“Card” Scene

- Noise increases with higher ISO
- Noise increases with smaller sensor
- JPEG lower noise, more accurate using CFA
- Scene content does change model (undersampling)

Summary

- PDF-based models are compact, understandable, and useful
  - Model can be treated as an image
  - Even 8-bit (256x256) models are effective
- Efficiently computed even from a single image
  - Scene content affects probability model, but not application
  - Accounting for CFA improves model quality
- The C++ source code for **`errpdf`** will be freely available