CSE 659A: Advances in Computer Vision

Spring 2020: T-R: 1:00-2:20pm @ Zoom
Instructor: Ayan Chakrabarti (ayan@wustl.edu).
http://www.cse.wustl.edu/~ayan/courses/cse659a/
April 14, 2020

BETTER NOISE PERFORMANCE WITH FLASH

- Problem: we have limited light in the scene, which is why we have to trade-off between noise and blur.
- Solution: Add light. Use a flash!
- Works, but a flash changes the appearance and "mood" of the scene.
  - Shadows and shading are now based on the flash, not the natural ambient light source in the scene.

COURSE ADMIN

- Proposal feedback posted.
- Last class this Thursday. No classes next week!
  - Use time to work on Project 2 and HW 5.
- Computational Photograph Conference happening (virtually) at WashU!
https://iccp2020.engr.wustl.edu/

(Bottom) Flash image has a totally different "mood" than (Top) ambient image.
- But the ambient image is much noisier.
- Want the shading of the no-flash image, but the quality of the flash image.
Better Noise Performance with Flash

Eisemann and Durand, "Flash Photography Enhancement via Intrinsic Relighting," SIGGRAPH 2004

- Color = Chromaticity image.
- Large Scale = Smoothing by Bilateral Filtering of Grayscale Image.
- Detail = Grayscale - Large Scale
- Some care needed for points in shadow in flash image.

A problem with using regular flash is that it's intrusive: blinding flashes of light are never fun. But that's because we are capturing an image with a regular camera that filters out light outside the visible range.

Idea: Make your flash emit ultra-violet and infra-red light, so that it's not visible to people in the scene...

... and have a camera whose pixels are sensitive to those light wavelengths.

Then, take an image with and without this "dark" flash, and reconstruct.

Krishnan and Fergus, "Dark Flash Photography," SIGGRAPH 2009
Flash/no-flash: We are taking two images, and getting "more light" using a flash. But even without a flash, realize that taking multiple images corresponds to collecting "more light".

If I took a long exposure: low noise, but chance of motion blur.

If I took multiple images in quick succession, I am dividing that long exposure into multiple short exposures. Each image is blur free, but noisy. But they are multiple noisy measurements of the same image.

I could average all images and this would be the equivalent of a long exposure (but remember the analysis from HW1 in 559A).

But the scene is moving, so averaging would lead me back to motion blur.

Key Idea: If I account for motion between the different images, and use other images as a reference to guide denoising of one image.

This is often the idea of "video denoising" algorithms. But can involve multiple steps that are fragile.

Train a neural network!


- Train a neural network that takes a burst of images as input, and produces a single denoised image as output.
- But if we set up our network to directly predict the denoised image (or even their residuals), this requires the network to carry through linear intensity information throughout multiple layers.
- Instead, we will denoise each pixel in each image by taking a weighted average of its neighbors.
- And use a neural network to predict what these weights should be.
- Consider averaging in \(K \times K\) neighborhoods, the neural network outputs \(K^2\) numbers (corresponding to a smoothing kernel), for every pixel in every image.
- And then, we will average the individual denoised images.
- But this process will also account for motions within the different images in the burst. Which means the individual kernels need not be centered, express both smoothing and "translation".
HDRI MAGING

- If your camera is recording an 8-bit image, then there are only 256 distinct intensity levels.
- What happens if your scene actually has a very high dynamic range: either you try to map the brightest level to 256, in which case large dark regions appear "flat", are mapped to the same value.
- Or you clip intensities beyond some limited, in which case bright regions show up as saturated.
- Many modern cameras allow you to take multiple images at different exposure levels, and then combine these with post-processing to form a high-dynamic range image.

Source: Wikipedia Article on HDR Imaging
**HDR IMAGING**

- But standard HDR imaging requires taking multiple images, need scene and camera to be static.
- Can we do this in a single shot.
- Idea 1: "spatially multiplex" different exposures.
- Remember, we are already putting different color filters in front of each pixel.
- But usually, if you select a red filter, you try to pass through as much light in the "red wavelengths" as possible.
- Instead, you could attenuate some pixels more (block more light), to simulate a shorter exposure, scale down its intensities.
- Kind of like having multiple exposures multiplexed in the same image.

---


---

**Hirakawa and Simon, "Single-Shot High Dynamic Range Imaging with Conventional Camera Hardware," ICCV 2011**

- In fact, you can use the relative transmission (between R,G,B) of the filters in regular cameras to exploit this as well, by being 'dynamic range' aware during the interpolation step.
- Get even better separation by putting a filter on the lens (which makes one color channel at all locations have lower exposure). No change in hardware needed!

---

**Le -to- Right: Regular Camera + Regular interpolation, Regular Camera + HDR Interpolation, Camera with Filter + HDR Interpolation.**
**HDR IMAGING**

- Regular quantization = Equally spaced bins between 0 and max-intensity.
- Proposed quantization: Make the intensity "wrap back" to zero, once it crosses a threshold.
- Then reason about intensities of neighboring pixels to figure out how many times each measured intensity has been "wrapped".

**COMPUTATIONAL PHOTOGRAPHY**

- So, these help mitigate noise (as well as effects from clipping for HDR).
- Next, we switch to a different form of degradation: Blur.
- Earlier in the course, we talked about "blind motion deblurring".
- Much more ill-posed than deconvolution, because you also need to estimate the blur kernel.
- But even there, we assumed a single blur kernel for the image.
- In reality, blur will be spatially varying.

**Motion Blur**

- Different objects will move at different speeds.
- Even if they are moving at the same speed, if they are at different depths, the "projected image speed" will be different.

**Defocus Blur**

- In this case, the blur will be different at each pixel based on depth.

**DEPTH FROM DEFOCUS**

- The defocus blur at each pixel depends on the depth of the corresponding point.
- If we could estimate the degree of defocus blur at every point, then we could get a depth map.
- Once we had that, we could deblur the image by using per-pixel defocus kernels.
- But defocus depth estimation is even harder than motion blur estimation (even when it’s not spatially varying)!

- Motion blurred images often have repeated edges, making them "look" un-natural to a prior. Defocus blur softens edges, have to decide if the image is blurred, or if the objects themselves have soft gradients.
DEPTH FROM DEFOCUS

Solution 1: Take multiple images with different focal settings.

\[ y_1[n] = (x \ast k_{1,d})[n], \quad y_2[n] = (x \ast k_{2,d})[n], \quad \ldots \]

- Typically, you will know what the blur kernel \( k_{d,i} \) is for a given depth value \( d \) for a focal setting \( i \) (from calibrating your camera).
- Now you have multiple measurements in which the depth \( d \) and sharp image \( x \) doesn’t change (assumes scene is static).
- Can use this to estimate depth and \( x \).

Ver 1: Focal Sweep

- Just take multiple images by changing the focal distance to \( f_1, f_2, f_3, \ldots \).
- For every point, see in which of the “focal stack” of images the pixel is the sharpest. (Compute gradient magnitude, do an arg-max).
- First approximation: set the depth \( d \) at each pixel to the focal distance of its sharpest observation.
- The problem: the way focusing works on most lenses, you can’t change the focal distance without moving the aperture with respect to the sensor plane.
- In other words, for projection equation \( x = kX/Z \), changing the focal distance also changes \( k \).
- So your images are also scaled in a depth-dependent way, and so this simple method won’t work.

Ver 2: Keep the focal distance the same, but change the aperture size.

- Now your images are all aligned.
- The size of the blur kernel is proportional to aperture size, and to distance of the point from the focal plane.
  \[ r \propto A(d - f)/f \]
- \( r \) is the radius of the blur kernel, \( A \) is the aperture radius which is changing across images, \( f \) is the focal distance that is fixed.
- So for every candidate depth value, you know what the different blur kernels will be for a sequences of images captured with aperture sizes \( A_1, A_2, \ldots \).
- Use this to estimate \( x[n] \) and \( d[n] \).
- And infact, most cameras already let you take a “burst” of images by changing the aperture size: as a way of doing HDR. Instead, we use it for depth estimation and deblurring.

Things get a little complicated though, because you also need to handle occlusions.

Hasinoff & Kutulakos, A Layer-Based Restoration Framework for Variable-Aperture Photography, ICCV 2007

Solution: Model the “latent” image as segmented into a collection of a finite \( K \) “layers”.

Each layer is at a distance \( d_k \), which induces a blur kernel \( B_k \) (for a specific aperture).

It’s appearance is given by an image \( L_k[n] \), but also with a mask \( A_k[n] \).

The sharp image \( X[n] \) is formed by taking the value of \( L_k[n] \) from the layer \( k \) with lowest depth, where

\[ A_k[n] = 1. \]

\[ X[n] = \sum M_k[n]L_k[n] \]

\[ M_k[n] = A_k[n] \prod_{k' = k+1}^{K} (1 - A_{k'}[n]) \]

assuming \( d_1 > d_2 > d_3 > \ldots \)

For the blurry case, work with “blurred” versions of \( L_k \) and \( A_k \).
Estimate the depths, appearance, and masks for all layers. (Combination of blur estimation + segmentation).

Make initial depth estimates based on blur cues, then use an MRF model to get a clean segmentation.

Need to resolve an ambiguity: \( r \propto A(d - f) \). Here, negative radii mean the blur kernel is flipped, but a flipped circle is a circle. So you know \(|d - f|\), but not its sign. Reason about this using consistency with occlusions.

Then use estimated depth and segmentation to get the sharp image for each layer.

They also handle high-dynamic range in the scene. Allow synthetic refocusing afterwards.

So this works great, but like with any multi-image system, need the scene to be static.

Can we do this with a single shot.

So let’s go back and look at why defocus blur estimation is hard.