COURSE ADMIN

- Reminder: HW 4 due Today.
- Proposal for Project 2 due Sunday (April 7).
- Slides for your talk need to be uploaded by next Sunday (April 14).

COMPUTATIONAL PHOTOGRAPHY

- Last Time: We talked about cameras that help mitigate noise (as well as effects from clipping for HDR).
- This time: 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.
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.

Solution 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 sequence of images captured with aperture sizes \( A_1, A_2, \ldots \).
- Use this to estimate \( x[n] \) and \( d[n] \).
- And in fact, 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.

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

Things get a little complicated though, because you also need to handle occlusions.
Estimate the depths, appearance, and masks for all $K$ 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: $r \propto A(d - f)f$. 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.
Blur kernels are the aperture shape (a nice circle) scaled by distance of point from focal plane.

Because of this, a sharp edge in a blurred image looks like a soft edge in a sharp image.

This is fundamentally because the blur kernel in defocus blur is a uniform symmetric circle (the shape of the aperture). Motion blur was easier because the blur kernels were less 'regular'.

Solution: change the blur kernel ...

by placing a mask in the aperture. So now, the blur kernels will be scaled versions of the mask, rather than scaled circles.

Levin et al., 2007
Veeraraghavan et al., 2007
**DEPTH FROM DEFOCUS**

Levin et al., "Image and Depth from a Conventional Camera with a Coded Aperture," SIGGRAPH 2007

Step 1: Find a Code for the Mask

- Take a dataset of natural images.
- Blur them with a bunch of randomly generated masks.
- Fit the gradients in images blurred by that mask to a probability distribution.
- Pick the mask where the distribution of blurred images is maximally different from that of sharp images (by KL divergence).

Step 2: Fabricate Mask, put it inside lens, take an image

Step 3: Estimate a depth-map

- For a discrete set of depth values \(d_1, d_2, d_3\), you have blur kernels \(k_1, k_2, k_3\).
- Given an observed image \(y[n]\), how do you decide which blur kernel \(k_i\) is the best explanation?
- For each \(i\):

  \[
  x_i[n] = \arg \min ||y[n] - (x_i * k_i)[n]||^2 + \lambda R(x_i)
  \]

  Get a regularized de-blurred estimate, assuming each kernel.

  \[
  E_i[n] = ||(x_i * k_i)[n] - y||^2
  \]

  Re-blur and measure error to observation.

- \(d[n] = \arg \min_i E_i[n]\)
- Then use an MRF to clean-up \(d[n]\).
- These guys also allow for incorporating user input.

Once you have depth map, \(d[n]\), you can deconvolve \(y[n]\) to get an all-sharp image.

- Given \(d[n]\) and \(x[n]\), you can synthetically simulate capture with different focal lengths.
So having a "binary mask" works better than a normal circular aperture.
But still needs user guidance for getting accurate depth maps.
Worse still: by putting in a mask, we're cutting off light. For the design we just saw, that kills almost 60% of the light going through the aperture.
Let's try to see if we can use a different property, or prior, of natural image gradients ... in color images.

But if I look at gradient profiles across color channels, they have the same shape (scaled by the difference in color-specific contrasts).

Chakrabarti and Zickler, "Depth and Deblurring from a Spectrally-varying Depth-of-Field", ECCV 2012.

We can encode this as a prior.
We can encode this as a prior.

Now all we need is an aperture code that "disrupts" this prior.

Pick a color filter that transmits red and blue, and filters out green.

Cut it into a circular shape with a 'hole' in the middle, and place it in your lens.
This makes the effective aperture size for the red and blue channels ... larger than that for green.

And so while a regular aperture would keep the rank-1 property of color gradient profiles

This modified aperture disrupts it.
Consider a candidate set of depths and their respective blur kernels. For each, try to apply a correction kernel for the green channel (instead of doing deconvolution).

Use an MRF to clean up the results.
DEPTH FROM DEFOCUS


Estimation

Now you can use the depth map to deblur and get an all sharp image.

\[
\begin{pmatrix}
1
\end{pmatrix}
\begin{pmatrix}
\text{Depth Map}
\end{pmatrix}
= \begin{pmatrix}
\text{Sharp Image}
\end{pmatrix}
\]

• And then use this for synthetic refocusing.

CODED APERTURE PHOTOGRAPHY

Zhou and Nayar, What are Good Apertures for Defocus Deblurring?, ICCP 2009

• We’ve talked about using coded apertures to make the “blind deconvolution” or blur estimation problem easier.
• But you can also think of using coded apertures that make the ‘non-blind deconvolution’ problem easier.
• Assume that you had other means to estimate depth, what should your blur kernel be so that it’s easiest to invert.
• Remember, in the Fourier domain

\[
y[n] = (x * k)[n] \rightarrow Y[f] = X[f]K[f]
\]

Try to find a blur kernel so that \(K[f]\) doesn’t become too small for any \(f\).

This is magnitude of \(X[f]\), so it takes into account the fact that different codes are cutting out different amounts of light (hence different values of \(X[0]\)).

CODED APERTURE PHOTOGRAPHY

Zhou and Nayar, What are Good Apertures for Defocus Deblurring?, ICCP 2009

• Can be used to undo very severe defocus blur.
So we saw that you can change the blur kernel for defocus to make estimation and inversion easier. You change the blur kernel by putting a mask in the aperture.

Let’s say we want to do the same for motion: have motion blur kernels that are easier to invert. How do you “mask” a motion blur kernel?

Defocus Blur: Open Aperture at some radius → Masked Aperture at same radius
Motion Blur: Open shutter for exposure time → Opening and closing the shutter according to some code ... while your sensor is still integrating light.


But this gives you a better way of inverting blur. But you still need a good way of estimating the blur kernel. For camera-shake, often times we can use a single blur kernel for the entire image (talked about this before). But now, let’s consider the problem of subject motion blur. Objects in your scene are moving, and each pixel can have different blur.

One advantage for the subject motion case, we can assume our blur kernels are “simpler”. Camera motion can be arbitrary because your hands are shaking and small movements in the camera can correspond to a large scaled version of your camera trajectory.

But for subject motion, blur kernels that are apparent in an image for an object that’s far away, can often be approximated by linear motion. I.e., the blur kernel is a line of some length at some orientation.

So this cuts down the space of possible blur kernels to say a discretized set of lengths and orientations.

We can try doing blur estimation from a regular camera as sort of classification among this discrete set. Can get somewhat decent results.

Chakrabarti et al., "Analyzing Spatially Varying Blur", CVPR 2010
The other option, capture images in a way so that the blur kernel is no longer spatially varying, even if objects are moving in different ways.

How do you do this?

By moving the camera during capture! "Optimized Camera Shake"

Can't undo subject motion blur, but can ensure that objects moving at different speeds have the same approximate blur kernel. And then deconvolve with that common kernel.

Assume all motion is along one dimension (say horizontal), is linear, but at arbitrary speeds.

Let’s analyze the problem in that dimension: Consider a 1D horizontal slice of the image begin formed on your camera.

And see how it changes through time. So plot $I(x, t)$.

Realize that your observed blurry image is $\int I(x, t)dt$, i.e., by integrating along the time dimension.
Now, if your camera was also moving, with its center given by $f(t)$, then you'd get a new image $I(x + f(t), t)$.

But you can choose your camera motion $f(t)$ in so that $\int I(x + f(t), t) dt$ is approximately the same, no matter what speed each point is moving at.

You get this by assuming that $I(x, t) = I(x + s(x)t)$, i.e., each point has linear motion with velocity $s(x)$.

With camera motion $I'(x, t) = I(x + s(x)t + f(t))$

Your observed image is $\int I'(x, t) dt = I * (\int \delta(s(x)t + f(t)) dt)$.

To make sure the blur kernel is approximately the same (upto translation) irrespective of $s(x)$, make $f(t) \propto t^2$, i.e., a parabola.
MOTION BLUR INVARIANCE

Levin et al., "Motion-invariant Photography", SIGGRAPH 2008

Figure 8: Deblurring results. Top row: image from a static camera. Bottom row: deblurring results using the image from our parabolic camera.