COURSE ADMIN

The Course after COVID

- All classes from now on will be on Zoom.
  - Recordings should be available on Canvas. Know that it might take a while for Zoom to process/encode the recordings.
- Will be posting slides to the course website as usual.
- No student presentations.
  Due to delays, you may not have enough time to work on them and the second project. Student presentations on zoom will also likely be hard.
- Total of HW + Projects will be scaled up as 100 / 85.
  If you switch to P/F, P threshold is 70 after scaling.
- Updated submission deadlines (see course website).

Any questions?

MOTION MAGNIFICATION

- The task: "magnify" motions.
- Don't make them faster, rather exaggerate small movements.

Input

Output
Initial methods were complex: estimate optical flow, amplify flow, warp image, use inpainting.

Liu et al., Motion Magnification, SIGGRAPH 2005.

But these methods are too fragile.
Instead, let’s go back and analyze what motion magnification is.

At time $t$, you had an image $f[x, y]$.

At time $t+1$, let’s say the entire image moved coherently, and you have $f[x + \Delta_x, y + \Delta_y]$.

To amplify this motion by 2, you would want to output $f[x + 2\Delta_x, y + 2\Delta_y]$.

Now, let’s say your image was a complex sinusoid at one frequency in one dimension.

$$f[x] = \exp(-j\omega x)$$

Remember, translation is a change in phase:

$$f[x + \Delta_x] = \exp(-j\omega(x + \Delta_x)) = f[x] \exp(-j\omega\Delta_x)$$

So translated image is at the same frequency, but with a different phase.

If you only had a sinusoid at a single frequency as your image, you could compute the phase at each time step, and amplify the change in phases.

Wadhwa et al., Phase-Based Video Motion Processing, SIGGRAPH 2013.

Use a complex steerable pyramid to decompose each frame.
- Each coefficient in this pyramid is a complex number corresponding to responses to sin and cos filters at different frequencies.
- Now, for the same coefficient, track the change in phase across time (i.e., over different frames).
- Amplify these phase differences, and reconstruct the motion.

Wadhwa et al., Phase-Based Video Motion Processing, SIGGRAPH 2013.
MOTION MAGNIFICATION

You can also do different kinds of smoothing on the phase time derivatives to suppress noise.
If your motion is oscillatory, individual phase values as a function of time will also be periodic. You can choose to only amplify motions that repeat at certain frequencies.

OTHER IMAGE PROCESSING APPLICATIONS

Deblurring

- We have talked about “non-blind” de-convolution before.
  \[ y[n] = (x * k)[n] + \epsilon[n] \]
- \( y \) is observed blurry image.
- \( x \) is true sharp image.
- \( k \) is blur kernel.
- \( \epsilon \) is noise. Often assumed Gaussian iid.

We have talked about recovering \( x \) from \( y \), if we know the blur kernel \( k \)
  \[ x = \arg \min_{x} \sum_{n} |y[n] - (x * k)[n]|^2 + R(x) \]
- But what if we don’t know \( k \) ?

Deblurring

\[ y[n] = (x * k)[n] + \epsilon[n] \]

Estimate both \( x \) and \( k \):
  \[ x, k = \arg \min_{x, k} \sum_{n} |y[n] - (x * k)[n]|^2 + R_1(x) + R_2(k) \]
- Now we have to solve for both \( x \) and \( k \). Priors / Regularizers on both \( x \) and \( k \).
- Assume \( k \) is unknown, but constant across the image. (There are versions of this problem where the blur is spatially varying, and the observation model is not a convolution).
- If we apply this for motion deblurring, typical \( k \)s look like motion trajectories.
Deblurring

\[ y[n] = (x \ast k[n]) + \epsilon[n] \]

Estimate both \( x \) and \( k \):

\[ x, k = \operatorname{arg\,min}_{x,k} \sum_n |y[n] - (x \ast k[n])|^2 + R_1(x) + R_2(k) \]

- \( R_2(k) \) is typically defined to constrain \( k[n] > 0, \sum_n k[n] = 1 \), and say \( k[n] \) is sparse (e.g., \( \sum_n |k[n]| \)).
- But the problem is that we need much stronger priors on \( x \).
- The above equation has a trivial solution: \( x = y, k = \delta[n] \) (1 at the center, 0 everywhere else).
- With typical image priors, \( R(y) \leq R(x) \). This is because \( y \) is a blurry image, and has gradients of smaller magnitudes.
- Most priors we’ve seen so far say gradients in a natural image should be small.

Stronger Image Priors: Internal Patch Recurrence


Key Idea: This internal recurrence is broken when your image is degraded.


Turn this recurrence property into a prior on our sharp image.
OTHER IMAGE PROCESSING APPLICATIONS


\[
 x, k = \arg\min_{x, k} \sum_n |y[n] - (x \ast k)[n]|^2 + R_1(x) + R_2(k)
\]

\[
 R_1(x) = \rho(x, x^a)
\]

- Where \(x^a\) represents the image \(x\) downscaled by a factor \(\alpha\).
- Basically, \(\rho\) learns a prior distribution \(\rho\) over patches in sharp images from \(x^a\).
- And \(\rho\) is the negative log-likelihood of this prior applied to patches in \(x\).
- Let \(\{R_i x^a\}\) correspond to all \(7 \times 7\) patches cropped for \(x^a\) (\(R_i\) is a linear matrix that “crops out” the \(i^{th}\) patch).
- Define \(p(x')\) as a probability distribution on a \(7 \times 7\) patch as

\[
p(x') \propto \exp\left(-\frac{1}{2h^2} \|x' - R_i x^a\|^2\right)
\]

- This is a ‘kernel’ density estimate with Gaussian kernel with bandwidth \(h\).
- Can think of this as a GMM, with one component each centered at every patch in \(x^a\), equal mixing probabilities for all components, and all with co-variance \(h^2 I\).

\[
x, k = \arg\min_{x, k} \sum_n |y[n] - (x \ast k)[n]|^2 + R_1(x) + R_2(k)
\]

\[
 R_1(x) = \rho(x, x^a)
\]

- Now let \(\{Q_j x\}\) denote the set of all (again \(7 \times 7\)) patches cropped from \(x\).

\[
 R_1(x) = -\sum_j \log \rho(Q_j x)
\]

\[
 R_1(x) = -\sum_j \sum_i \exp\left(-\frac{1}{2h^2} \|x' - R_i x^a\|^2\right)
\]

This is a highly non-convex objective.

Iterative Minimization

Begin with initial estimate of \(x = y\) and \(k = \delta\). And iterate between:

- Construct \(x^a\) by downsampling \(x\).
- Minimize objective with respect to \(x\) holding \(x^a\) and \(k\) fixed.
- Minimize objective with respect to \(k\) holding \(x\) fixed.

In practice, they do this in a coarse to fine manner: Solving the deblurring problem from downsampled \(y\) to estimate downsampled \(x\) and \(k\), and up-sampling after convergence at each scale.

Once they have an estimate of \(k\), use a regular image prior for non-blind deconvolution to get \(x\).
DEHAZING

Two Effects
- Scatter away a portion of the light coming towards the camera from object.
- Scatter light coming directly from light source towards camera.

And both these effects will be greater the farther the object is away from the camera: more scattering particles in between.

Two Effects
- Scatter away a portion of the light coming towards the camera from object.
- Scatter light coming directly from light source towards camera.

And both these effects will be greater the farther the object is away from the camera: more scattering particles in between.

True if light was traveling in vacuum.

But the atmosphere (or other media: say you are underwater) will also interact with the light.
DEHAZING

• The problem, estimate the clean image given a hazy observation.
• Simplified observation model:

\[ Y[n] = t[n] L[n] + (1 - t[n]) A \]

• \( L[n] \) is the haze-free image. \( Y[n] \) is the observation. Both are 3-channel.
• \( t[n] \) is a transmission coefficient that is dependent on depth. Assume \( t[n] = \exp(-\beta Z[n]) \).
• \( A \) is a common 3-channel vector for the entire image, called “Airlight”. Color of the haze (depends on the light color).
• Estimate \( L[n] \) given \( Y[n] \). Ill-posed because you don’t know \( t[n] \) or \( A \).

DEHAZING

Bahat and Irani, Blind Dehazing Using Internal Patch Recurrence, ICCP 2016.

\[ Y[n] = t[n] L[n] + (1 - t[n]) A \]

• Use patch-recurrence. Assume \( L[n] \) has similar patches which appear different in \( Y[n] \) because of the haze.

DEHAZING

\[ Y[n] = t[n] L[n] + (1 - t[n]) A \]

• Assume you have two patches \( P_1 \) and \( P_2 \) from \( Y \), which share the same \( L \).
• Further assume that because \( t[n] \) is smooth, it is different for the two patches, but constant inside each patch.

\[
\begin{align*}
P_1[n] &= t_1 L[n] + (1 - t_1) A \\
P_2[n] &= t_2 L[n] + (1 - t_2) A
\end{align*}
\]

• You can then compute the “ratio” between \( t_1 \) and \( t_2 \) as the ratio of the standard deviations of \( P_1 \) and \( P_2 \).
• This is because the effect of \( A \) only changes the mean.
• You also have that

\[ P_2[n] - A = \frac{t_2}{t_1} (P_1[n] - A) \]

• Can solve for \( A \).