CSE 559A: Computer Vision

Fall 2019: T-R: 11:30-12:50pm @ Hillman 60

Instructor: Ayan Chakrabarti (ayan@wustl.edu).
Course Staff: Arushee Agrawal, Annie Lee, Jiahao Li, Xiaochen Zhou

http://www.cse.wustl.edu/~ayan/courses/cse559a/

Aug 29, 2019
All information @ http://www.cse.wustl.edu/~ayan/courses/cse559a/

- Read through course website
  - Syllabus
  - Problem Set and Resource Section
  - Late Policies
    - Collaboration & Academic Honesty Policies
- Join Piazza
- Review Pre-req Slides
- Install Anaconda, LaTeX, Git
- Submit your Public Key
- Submit Problem Set 0
Office Hours

- Arushee & Annie: Tuesdays, 10am-11am
- Jiahao & Xiaochen: Thursdays, 5pm-6pm

For now, office hours will be held in the collaboration space in Jolley 217.

- Starts today evening!
• \( E(x, y, t) \): Light energy, \textit{per unit area per unit time}, arriving at point \((x, y)\) at time \(t\)
  - Here, \(x, y\) are real numbers (in meters) denoting actual position on the sensor plane.

• \( I[n_x, n_y] \): Intensity measured by the sensor element at grid location \(n_x, n_y\)
  - Here, \(n_x, n_y\) are integers, indexing pixel location.
  - Note the convention \([\cdot]\) for "indexing", while \((\cdot)\) for \(E\) because it is a function.

• \( p(x, y) \): is a sensitivity function for a pixel, assuming \((0,0)\) is the center of the pixel.
  - \(p(\cdot, \cdot)\) is ideally 1 inside pixel (for example, within \([-W/2, -H/2]\) to \([W/2, H/2]\)), and 0 outside. But may have attenuation at boundaries.
  - \(\bar{x}_{n_x}, \bar{y}_{n_y}\) is the location (in meters) of the center of the sensor element
  - \(p(x - \bar{x}_{n_x}, y - \bar{y}_{n_y})\) is the sensitivity function for the pixel at \([n_x, n_y]\), "centered" at \((\bar{x}_{n_x}, \bar{y}_{n_y})\).
- \( E(x, y, t) \): Light energy, *per unit area per unit time*, arriving at point \((x, y)\) at time \(t\)
  - Here, \(x, y\) are real numbers (in meters) denoting actual position on the sensor plane.
- \( I[n_x, n_y] \): Intensity measured by the sensor element at grid location \(n_x, n_y\)
  - Here, \(n_x, n_y\) are integers, indexing pixel location.
  - Note the convention \([\cdot]\) for "indexing", while \((\cdot)\) for \(E\) because it is a function.
- \( p(x, y) \): is a sensitivity function for a pixel, assuming \((0,0)\) is the center of the pixel.
  - \(p(\cdot, \cdot)\) is ideally 1 inside pixel (for example, within \([-W/2,-H/2]\) to \([W/2,H/2]\)), and 0 outside. But may have attenuation at boundaries.
  - \(\bar{x}_{n_x}, \bar{y}_{n_y}\) is the location (in meters) of the center of the sensor element
  - \(p(x - \bar{x}_{n_x}, y - \bar{y}_{n_y})\) is the sensitivity function for the pixel at \([n_x, n_y]\), "centered" at \((\bar{x}_{n_x}, \bar{y}_{n_y})\).
- Defining \(q\) as the "quantum efficiency" of the sensor: Ratio of Light Energy to Charge/Voltage
  - \(\int E(x, y, t) \ p(x - \bar{x}_{n_x}, y - \bar{y}_{n_y}) \ q \ dx \ dy\)
    - Rate at which charge/voltage increases in sensor element \([n_x, n_y]\) at time \(t\).
- An image capture involves "exposing" the image for an interval \(T\) (seconds).
- So the total intensity is going to involve integrating the charge/voltage rate over that interval.
\[ I[n_x, n_y] = \int_{t=0}^{T} \left[ \int E(x, y, t) \ p(x - \bar{x}_{n_x}, y - \bar{y}_{n_y}) \ q \ dx \ dy \right] \ dt \]

- \( n_x, n_y \) are integers indexing pixels in image array.
- \((x, y)\) is spatial location
- \( I[n_x, n_y] \) is recorded pixel intensity.
- \( E(x, y, t) \) is light "power" per unit area incident at location \((x, y)\) on the sensor plane at time \( t \)
- \((\bar{x}_{n_x}, \bar{y}_{n_y})\) is the "center" spatial location of the pixel / sensor element at \([n_x, n_y]\).
- \( p(x, y) \) is spatial sensitivity of the sensor (might be lower near boundaries, etc.)
- \( q \) is quantum efficiency of the sensor (photons/energy to charge/voltage)
- \( T \) is the duration of the exposure interval.

CCD/CMOS sensors measure total energy or "count photons" that arrived during exposure.
$I[n_x, n_y] = \int_{t=0}^{T} \left[ \int E(x, y, t) \ p(x - \bar{x}_{n_x}, y - \bar{y}_{n_y}) \ q \ dx \ dy \right] \ dt$

$I[n_x, n_y]$ is the *ideal unquantized* pixel intensity
SENSOR

\[ I^0[n_x, n_y] = \int_{t=0}^{T} \left[ \int E(x, y, t) \ p(x - \bar{x}_{n_x}, y - \bar{y}_{n_y}) \ q \ dx \ dy \right] dt \]

\[ I \leftarrow I^0 \]

SHOT NOISE

- Caused by uncertainty in photon arrival
- Actual number of photons \( K \) is a discrete random variable with Poisson distribution
  
  \[ P(K = k) = \frac{\lambda^k e^{-\lambda}}{k!} \]
  
  - \( \lambda \) is the "expected" number of photons. In our case, \( \propto I^0 \)
SHOT NOISE

- Property of Poisson distribution: Mean and Variance both equal
- Often, shot noise is modeled with additive Gaussian noise with "signal dependent" variance:

\[ I \leftarrow I^0 + \sqrt{I^0} \epsilon_1 \]

where \( \epsilon \sim \mathcal{N}(0, 1) \) (Gaussian random noise with mean 0, variance 1).

\[ I \sim \mathcal{N}(I^0, I^0) \]
**SENSOR**

\[ I^0[n_x, n_y] = \int_{t=0}^{T} \left[ \int E(x, y, t) \ p(x - \bar{x}_{n_x}, y - \bar{y}_{n_y}) \ q \ dx \ dy \right] dt \]

\[ I \leftarrow I^0 + \sqrt{I^0} \ \epsilon_1 \]

**AMPLIFICATION & ADDITIVE NOISE**

- Signal amplified by gain \( g \) before digitization. Based on ISO (higher \( g \) for higher ISO).
- Some signal-independent Gaussian noise added before and after amplification.

\[ I \leftarrow g \times (I^0 + \sqrt{I^0} \ \epsilon_1 + \sigma_{2a} \epsilon_{2a}) + \sigma_{2b} \epsilon_{2b} \]

where \( \sigma_{2a} \) and \( \sigma_{2b} \) are parameters (lower for high quality sensors), and \( \epsilon_1, \epsilon_{2a}, \epsilon_{2b} \) are \( \mathcal{N}(0, 1) \) noise variables, all independent.

\[ I \leftarrow gI^0 + g \sqrt{I^0} \ \epsilon_1 + \sqrt{\left(g^2 \sigma_{2a}^2 + \sigma_{2b}^2\right)} \ \epsilon_2 \]

Amplified Signal  
Amplified Shot Noise  
Amplified and un-amplified additive noise
$I^0[n_x, n_y] = \int_{t=0}^{T} \left[ \int E(x, y, t) \ p(x - \bar{x}_{n_x}, y - \bar{y}_{n_y}) \ q \ dx \ dy \right] \ dt$

$I \leftarrow gI^0 + g\sqrt{I^0} \ e_1 + \sqrt{g^2\sigma^2 + \sigma^2} \ e_2$

**DIGITIZATION**

- Final step is rounding and clipping (by an analog to digital converter)

$I \leftarrow \text{Round}\left( gI^0 + g\sqrt{I^0} \ e_1 + \sqrt{g^2\sigma^2 + \sigma^2} \ e_2 \right)$

$I = \min\left( I_{\text{max}}, \ \text{Round}\left( gI^0 + g\sqrt{I^0} \ e_1 + \sqrt{g^2\sigma^2 + \sigma^2} \ e_2 \right) \right)$
**SENSOR**

\[
I^0[n_x, n_y] = \int_{t=0}^{T} \left[ \int E(x, y, t) \ p(x - \bar{x}_{n_x}, y - \bar{y}_{n_y}) \ q \ dx \ dy \right] \ dt
\]

\[
I = \min\left( I_{\text{max}}, \ \text{Round}\left( gI^0 + g\sqrt{I^0} \ \epsilon_1 + \sqrt{(g^2 \sigma_{2a}^2 + \sigma_{2b}^2) \ \epsilon_2} \right) \right)
\]

ignoring sensor saturation, dark current, ...

**WHY STUDY THIS ?**

- To understand the degradation process of noise (if we want to denoise / recover \( I^0 \) from \( I \)).
- To prevent degradation during capture, because we control exposure time \( T \) and ISO / gain \( g \).
- To understand the different trade-offs for loss of information from noise, rounding, and clipping.
\[ I^0[n_x, n_y] = \int_{t=0}^{T} \left[ \int E(x, y, t) p(x - \bar{x}_{n_x}, y - \bar{y}_{n_y}) q \, dx \, dy \right] dt \]

\[ I = \min \left( I_{\max}, \, \text{Round} \left( gI^0 + gI^0 \epsilon_1 + \sqrt{(g^2\sigma^2_{2a} + \sigma^2_{2b})} \epsilon_2 \right) \right) \]

**ROUNDING VS CLIPPING**

Ignoring noise, what is the optimal \( g \) for a given \( I^0[n_x, n_y] \)?

- Keep \( g \) low so that most values of \( gI^0[n_x, n_y] \) are below \( I_{\max} \).
- But if \( g \) is too low, a lot of the variation will get rounded to the same value.
Increasing $g$

Data from Sam Hasinoff
Note that here, our 'ideal' intensity is $gI^0$, everything else is noise.

**LIGHT VS AMPLIFICATION**

Say we have chosen the optimal target values for the product $gI^0$. Is it better:

- To have a higher $g$ and lower magnitude $I^0$
- **To have a lower $g$ and higher magnitude $I^0$**
- Depends, based on $\sigma_{2a}, \sigma_{2b}$

**Additional Reading (if interested):**

So how do we increase $I^0$?

- Better sensors (higher $q$)
- Larger sensor elements: $p(\cdot, \cdot) > 0$ over a larger area.

But we've gone the other way: cameras stuff more 'megapixels' in smaller form factors.
SENSE

\[ I^0[n_x, n_y] = \int_{t=0}^{T} \left[ \int E(x, y, t) p(x - \bar{x}_{n_x}, y - \bar{y}_{n_y}) q \, dx \, dy \right] dt \]

Increase exposure time \( T \)?

- If scene is static and camera is stationary:
  - \( E(x, y, t) \) doesn't change with \( t \) \( \Rightarrow I^0 \propto T \)
- If scene is moving ...

![Diagram with a flower plant before and after motion blur]

Motion Blur
Increase $E(x, y, t)$ itself. How?

- Take pictures outdoors, or under brighter lights.
- Don't use a pinhole camera!
Small Pinhole: Focused beam, very little light.
Large Pinhole: More light, larger 'circle of confusion'
Solution: Put a lens to focus
Thin lens model (approximation)

More light & localization of rays to sensor plane but only for objects at the focusing distance $D$. 

\[
\frac{1}{D} + \frac{1}{f} = \frac{1}{\bar{f}}
\]

Distance of Sensor to Aperture

Focal length of Lens
Again, point projects to 'circle of confusion'
Circle of Confusion still proportional to aperture size.

Can still tradeoff amount of light with amount of blur in out of focus regions (move up or down "f-stops")

Better trade-off than pinhole camera.
Photographers think about these tradeoffs every time they take a shot

- Dynamic range and what part of the image should be well exposed (rounding and clipping)

- Choosing between:
  - ISO i.e. Gain & noise
  - Exposure Time & motion blur
  - F-stop i.e. aperture size & defocus blur
We left out at an important term in this equation: wavelength

\[
I^0[n_x, n_y] = \int_{t=0}^{T} \left[ \int E(x, y, t) p(x - \bar{x}_n, y - \bar{y}_n) q \, dx \, dy \right] dt
\]
COLOR

\[ I^0[n_x, n_y] = \int_{t=0}^{T} \left[ \int E(\lambda, x, y, t) \ p(x - \bar{x}_n, y - \bar{y}_n) \ q(\lambda) \ d\lambda \ dx \ dy \right] dt \]

- Light carries different amounts of power in different wavelengths
- \( E(\lambda, x, y, t) \) now refers to power per unit area per unit wavelength
  - In wavelength \( \lambda \), incident at \((x, y)\) at time \( t \)
  - Both spectral and spatial density function
- \( q(\lambda) \): Quantum efficiency also a function of wavelength
  - CMOS/CCD sensors are sensitive (have high \( q \)) across most of the visible spectrum
  - Actually extend to longer than visible wavelengths (near infra red)
  - Why cameras have NIR filter, to prevent NIR radiation from being 'superimposed' on the image

Q: But this measures 'total' power in all wavelengths. How do we measure color?

Ans: By putting a color filter in front of each sensor element.
COLOR

\[ I^0[n_x, n_y, c] = \int_{t=0}^{T} \left[ \int E(\lambda, x, y, t) \ p(x - \bar{x}_{n_x}, y - \bar{y}_{n_y}) \ \Pi_c(\lambda) \ q(\lambda) \ d\lambda \ dx \ dy \right] \ dt \]

for \( c \in \{R, G, B\} \)

- \( \Pi_c \) is the transmittance of a color filter for color channel \( c \)
- E.g., \( \Pi_R \) will transmit power in (be high for) wavelengths in the red part of the visible spectrum and attenuate power in (be low for) other wavelengths.
- Sometimes also called "color matching functions"

![Graph showing color matching functions](image-url)

\( \text{actually XYZ not RGB} \)
COLOR

\[
I^0[n_x, n_y, c] = \int_{t=0}^{T} \left[ \int E(\lambda, x, y, t) \ p(x - \bar{x}_{n_x}, y - \bar{y}_{n_y}) \ \Pi_c(\lambda) \ q(\lambda) \ d\lambda \ dx \ dy \right] dt
\]

for \( c \in \{R, G, B\} \)

- But we can only put one filter in front of each sensor element / pixel location.
- So color cameras "multiplex" color measurements: they measure a different color channel at each location.
- Usually in an alternating pattern called the Bayer pattern:

![Bayer Pattern](image)

- Note: a disadvantage is that color filters block light, so measured \( I^0 \) values are lower.
- That's why black and white / grayscale cameras are "faster" than color cameras.
Final steps in camera processing pipelines (except for some DSLR cameras shooting in RAW):

- **Filter Colors to Standard RGB:**
  - Cameras often use their own color filters $\Pi_c$.
  - Apply a linear transformation to map those measurements to standard RGB.
- **White-balance:** scale color channels to remove color cast from a non-neutral illuminant.
- **Tone-mapping:**
  - The simplest form is "gamma correction" (approximately raising each intensity to the power $(1/2.2)$)
  - Done based on standard developed for what old display devices expected
  - Fits the full set of measurable colors into the gamut that can be displayed / printed
  - Modern cameras often do more advanced processing (to make colors look vibrant)
- **Compression**

And that's how you get your PNG / JPEG images!

**Optional Additional Reading:** Szeliski Sec 2.3
OTHER EFFECTS

Other effects we did not talk about. E.g.,

- Real Lenses not thin lenses and have distortions:
  - Radial Distortion
  - Vignetting
  - Chromatic Aberration
OTHER EFFECTS

Other effects we did not talk about. E.g.,

Rolling Shutter: No explicit shutter but when pixels reset electronically (along scanlines)
NON-STANDARD CAMERAS

Place array of micro-lenses infront of sensor
Different pixels observe the scene "from a different angle"
Let you estimate depth, shift view point post-capture,
refocus post-capture
16 Different Camera Units
Fuse images to get lower noise, higher dynamic range, synthetically control focal distance and aperture size post-capture

No lens: lets the camera be much smaller, sensor can be placed on curved surfaces. Put a mask in the aperture: causes 'patterns' of confusion instead of circles of confusion. Create focused image computationally.
Coded Aperture Photography

NON-STANDARD CAMERAS

Synthetic Focus Post-capture

Coded Aperture Photography

Unbounded High Dynamic Range Photography using a Modulo Camera

Hang Zhao¹  Boxin Shi¹,³  Christy Fernandez-Cull²  Sai-Kit Yeung²  Ramesh Raskar¹

¹ MIT Media Lab  ²MIT Lincoln Lab  ³SUTD

ICCP 2015, Houston, TX  [Best Paper runner-up]

Intensity camera  Modulo camera  Recovered

Single-capture HDR  Multi-capture HDR: 8 bits + 8 bits = 16 bits
IMAGES

- Exist as 2-D (grayscale) or 3-D (color image) arrays

![2-D and 3-D images](image.png)

- Precision: uint8 (0-255), uint16(0-65535), Floating point (0-1)
  - We will often treat them as (positive) real numbers.

- Conventions:
  - \( I[n_x, n_y] \in \mathbb{R} \) (for a grayscale image), where \( n_x, n_y \in \mathbb{Z} \)
  - \( I[n_x, n_y, c] \in \mathbb{R} \) (for a color image)
  - \( I[n_x, n_y] \in \mathbb{R}^3 \) (for a color image)
  - \( I[n] \in \mathbb{R} \) or \( \in \mathbb{R}^3 \), where \( n \in \mathbb{Z}^2 \)

[numpy order: (H, W) or (H,W,3)]
NUMPY CONVENTIONS

- An image will exist as a numpy array when you load it, with dimensions (H,W) or (H,W,C)
- In numpy, you can recover sub-arrays by indexing into the main array.

```python
# Assume img is a 3D array of shape (H,W,3)
img2 = img[:, :, 0]  # Returns a 2D array of shape (H,W). Corresponds to the red channel.

img2 = img[:, :, 0:1]  # Returns the same thing, but now as a 3D array of shape (H,W,1).

img2 = img[0:10, 0:10, :]  # Returns array of shape (10,10,3). The top 10x10
# patch of the image (going from pixels 0 to **9**)

img2 = img[0:10:2, 0:10:2, :]  # Returns array of shape (5,5,3). Every alternate
# pixel in top 10x10 patch starting from 0,0.
```
When a number is omitted in the \([a:b:c]\) convention, \(a\) and \(b\) are assumed to be start or end of the image along that axis (depending on whether \(c\) is positive or negative), and \(c\) is assumed to be 1.

Indexing can also be used for assignment. \(\text{img2}\) should be the right shape of the "slice" corresponding to \(\text{img}[::2,::2,:]\).

Note that array variables are "pointers" to data. Assignment (without indexing) will not create a new copy of the data automatically.
NUMPY CONVENTIONS

- Reshaping: All of your data in arrays is stored sequentially, with the last dimension changing fastest.
  - For an array of size (H,W,3), $img[a,b,c+1]$ is stored right after $img[a,b,c]$, while $img[a,b+1,c]$ is stored 3 locations later.

```python
>>> print(v) # is a (3,3) array
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
>>> s = v.reshape((9))
>>> print(s)
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
>>> v2 = s.reshape( (3,3) )
>>> print(v2)
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

- `array.reshape()` takes a numpy tuple of the shape. The product of the elements of the tuple should match the total number of elements in the array. You can also leave upto one dimension as -1, in which case numpy will automatically calculate the value of that dimension.

```python
>>> s = v.reshape((-1))
```
NUMPY CONVENTIONS

- Simple shortcut: \( X[:, \text{np.newaxis}, :] \) is equivalent to \( X \text{. reshape((H, 1, W))} \) if \( X \) had shape (H,W).

IMPORTANT: Array reshapes is not the same as taking "transpose".

```python
>>> print(v)  # is a (3,2) array
array([[0, 1],
       [2, 3],
       [4, 5]])

>>> v2 = np.reshape((2, 3))

>>> print(v2)
[[0 1 2]
 [3 4 5]]
```

- Ok, so now, what are some of the operations we can perform on images?
POINT-WISE OPERATIONS

- \( Y[n] = h(X[n]) \)
- \( Y[n] = h(X_1[n], X_2[n], \ldots) \)
- \( Y[n] = h_n(X[n]) \) - Might vary based on location.
- \( h(\cdot) \) itself might be based on 'global statistics'
POINT-WISE OPERATIONS

\[ X[n] \in \mathbb{R}^3 \]
POINT-WISE OPERATIONS

\[ X[n] \in \mathbb{R}^3 \]

\[ Y[n] = \begin{bmatrix} 0.7 & 1.05 & 0.7 \end{bmatrix} X[n] \]

Linear Color Transforms
POINT-WISE OPERATIONS

\[ X[n] \in \mathbb{R}^3 \]

\[
Y[n] = \begin{bmatrix}
0.7 & 0.7 \\
1.05 &
\end{bmatrix} X[n]
\]

Linear Color Transforms
POINT-WISE OPERATIONS

\[ X[n] \in \mathbb{R}^3 \]

\[ Y[n] = \begin{bmatrix} 1.05 & 0.7 & 0.7 \end{bmatrix} X[n] \]

Linear Color Transforms
POINT-WISE OPERATIONS

\[ X[n] \in \mathbb{R}^3 \]

\[
Y[n] = \begin{bmatrix}
1/3 & 1/3 & 1/3 \\
1/3 & 1/3 & 1/3 \\
1/3 & 1/3 & 1/3 \\
\end{bmatrix} X[n]
\]

Linear Color Transforms
POINT-WISE OPERATIONS

\[ X[n] \in \mathbb{R}^3 \]

Linear Color Transforms

\[ Y[n] = 0.9 \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix} X[n] + 0.1 X[n] \]
POINT-WISE OPERATIONS

\( X[n] \in \mathbb{R}^3 \)

Linear Color Transforms

\[
Y[n] = 0.5 \begin{bmatrix}
1/3 & 1/3 & 1/3 \\
1/3 & 1/3 & 1/3 \\
1/3 & 1/3 & 1/3 \\
\end{bmatrix} X[n] + 0.5 X[n]
\]
POINT-WISE OPERATIONS

\[ X[n] \in \mathbb{R}^3 \]

Linear Color Transforms

\[
Y[n] = -1.0 \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix} X[n] + 2.0 X[n]
\]