Image Formation Notes

ELC4353
Spring 2026
Lecture by Professor Keith Schubert, Notes by Chris F Lin

1 Tuesday 01/20/2026- Introduction

TEXT: **Digital Image Processing, 4th North American ed**: Gonzalez & Woods
Topics:

- Overview of Imaging Systems and Techniques
- Spatial Processing
- Frequency Processing
- Reconstruction
- Morphological Processing
- Segmentation
- AI/ML

Optical illusions: our brains try to fill in spaces with shapes we believe should be there.

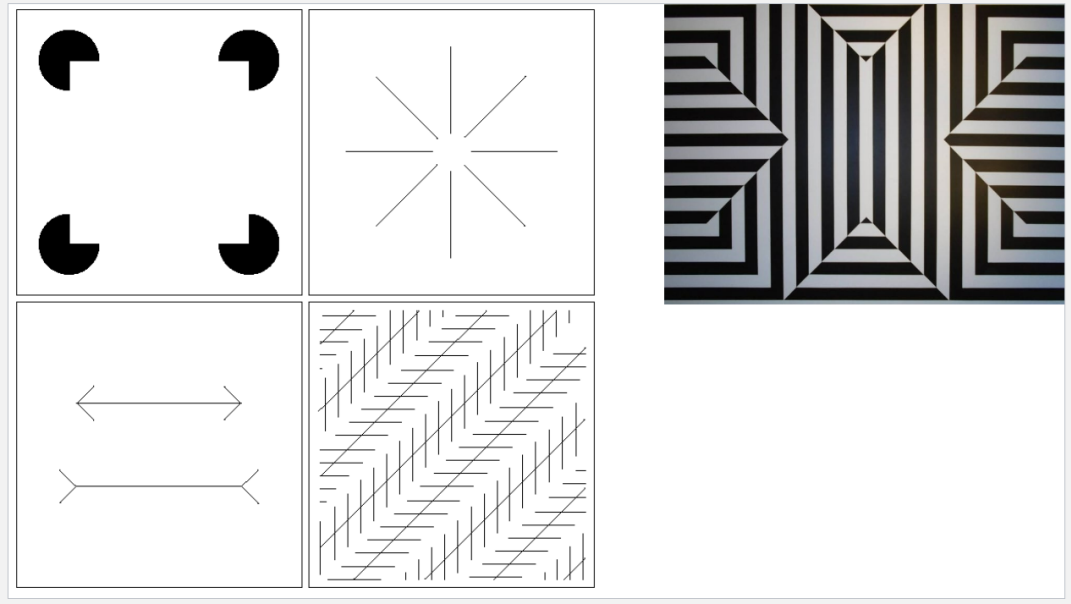


Figure 1: Illusions

How do we handle background gray colors vs general shape inside? Round shape and gray scale produces illusion of a rotation. Density of points can give appearance of grayscale. Technique

originated in 1800s, transmitting images via pointclouds. In general, contrasts may produce illusion that there is differences in intensity. Surrounding setups can manipulate interpretations.

Overall, visibile spectrum is incredibly small bandwith of total spectrum:

- ultraviolet
- violet: $0.4e-6$
- blue
- green: $0.5e-6$
- yellow: $0.6e-6$
- orange
- red: $0.7e-6$
- infared

We care about RGB because all colors we can see are compositions of the three colors. Dynamic range plays a big part in visualization. What you want needs to lie within the dynamic range. UV/IF, for humans, we have it in genetic code but not expressed so those cones don't exist in us; animals have this genome expressed so are able to see this extra range.
READ: Chapter 1

2 Thursday 01/22/2026- Image Formation

To talk about how computers make images, we use biology as inspiration; focus on retina, this is where processing occurs.
Majority of color detection occurs in *Fovea*; number of cones overly built-up here. Cones much more spread out (except the blind spot). Cones also used for more dense vision, higher resolution. See figure 2. Rods form an array, enable detection of lower light volume. Lower cone counts correlate with color-blindness. IE cats have significantly more rods, but less cones (only see 2 colors vs 3 colors). More rods, tradeoff is less resolution.
For printing, typically use CYMK vs RGB, otherwise too intense, too much darkenss buildup.
So why does the blind spot region have 0 rods or cones? Rods and cones are closer toards the lens. Need lots of blood and requires a cooling effect (blood serves double purpose). Rod/cones on top because it concentrates/intensifies the light; blood is on the backend because blood prevents/dims light.
Eye exams can provide give insight on overall health, simply due to the high blood sensitivity.
Visualization of "eye looking at a palm tree", the center C is the optical center of the lens. Similar triangles formed because light is not distorted at the center. From a computer stsandpoint, we know the size of the array at the back of the eye (or the detector). Reminder that the image formed is technically flipped upside down (fig 3). A great difficulties comes when we dont have perfect alignment with pixels. Some other terms:

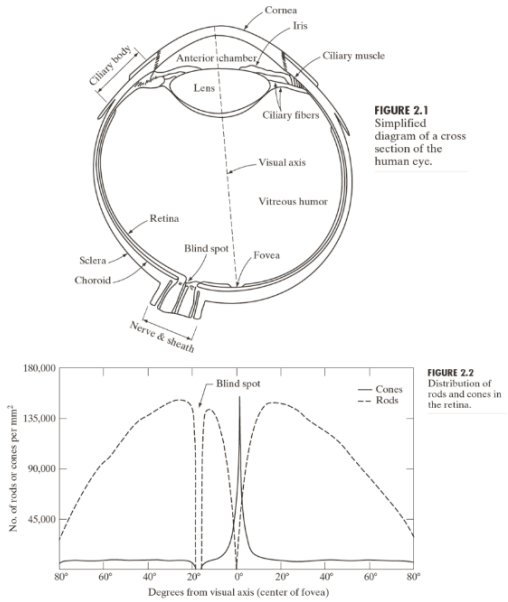


FIGURE 2.1 Simplified diagram of a cross section of the human eye.

FIGURE 2.2 Distribution of rods and cones in the retina.

Figure 2: Illusions

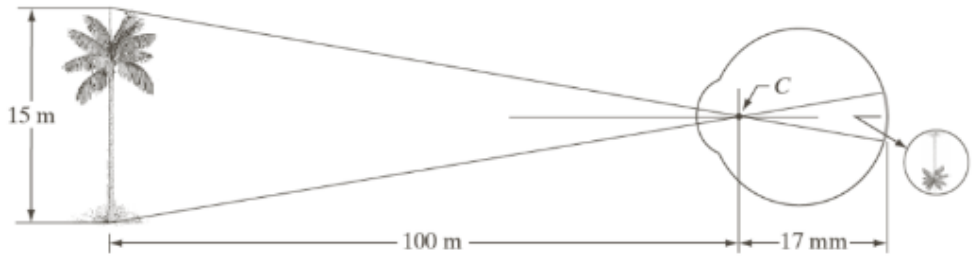


FIGURE 2.3 Graphical representation of the eye looking at a palm tree. Point *C* is the optical center of the lens.

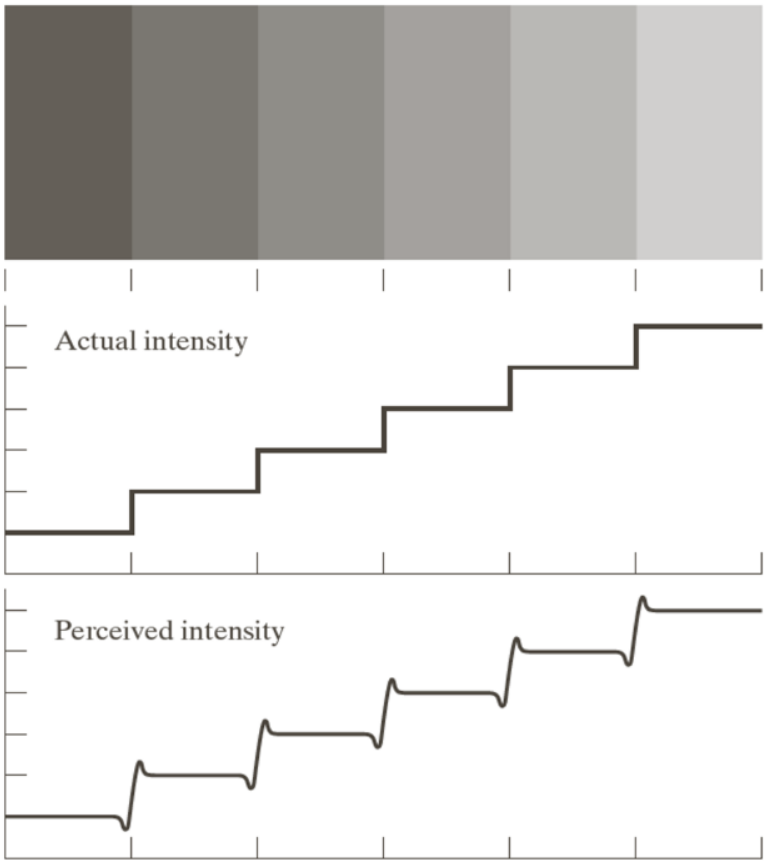
Figure 3: Illusions

- Scotopic: night vision; low-light version of eye detection
- Photopic: what your cones detect, big range. extends up to "glare limit".
- Basic tests can characterize brightness discrimination. Why is intensity test a circle and not just a square? If we use squares, we look directly at the corner. We are now testing a second thing (instead of just intensity); looking for corners too now. Now we're no longer only testing one thing.
- Weber ratio, function of intensity. Rods and cones have their own regions of intensity that they work in. Want more negative for better resolution. but as intensity goes down, only rods function in that region.

So why do we care about these things? Want to know what matters to us, so we know what we want to train towards, what we want to focus on when building our own systems.

Design algorithms that meet human needs.

If we compare actual intensity vs perceived intensity, actual intensity is constant. However, we want to see edges. So our "perceived" intensity will make it darker on one side, lighter on another to make the edges more crisp. this is basis of a *sharpening mask* 4. So how can we build images?



a
b
c

FIGURE 2.7
Illustration of the Mach band effect. Perceived intensity is not a simple function of actual intensity.

Figure 4: Illusions

Sensing material typically a (doped) semiconductor. We use semiconductors because responsive to photons due to energy transition range. Filters used to remove UV and helps lowering energy coming in. Black lights work because some materials react to UV light (which is emitted by black lights).
Another example, birds can detect pollen because pollen reflects heavily UV light. For the detector, housing unit is important to keep things separate. You can do things with a single pixel array (ie light/darkness detector).
Can also form a row/1D array (photocopier/scanner), or even a matrix. Matrix requires many wires so very complex, so typically not read out simultaneously. Often times only read out row or column at a time. Shutter allows stopping everything, removes motion blur.
When we digitize real object, we have 2 areas of discretization; spatial discretization and intensity discretiztion. There are differences in quality depending on change in gray-scale levels, size of pixels.
At a certain point, you hit saturation so can't receive any additionally. Impossible to fully restore, but can apply corrections and assumptions to fill in.
Regarding memory requirements, if our array is M by N, if each can register 2^k values, our system

requires a total of M x N x k bits.

READ: 2.1-2.3

3 Tuesday 01/27/2026- Local operations

Basic quantization by pixels; a fun way to it : splitting each of RGB into it's own plane (bitplanes; 1st bit represents first plane, 2nd bit is 2nd plane, etc). exists matlab function to extract bits

Determining 'regions' can vary; this is dependent on what we define to be "connected/adjacent" or connectivity/adjacency. Could be only adjacent, or also includes diagonal Will later discuss algorithms to identify connectivity. Will define two versions:

Definition 1 (Connectivity)**. 4-connectivity** is if directly adjacent pixels are highlighted. **8-connectivity** is 4-connectivity, expanding to include diagonals.

If we have a maze that has one entrance/ one exit, we can segment it into two sides. Therefore, we will always have two different regions, simply follow where the two sides (of the path) are always different colors (5).

Now, in matlab, we're going to code up a way to color the different regions, one per region (see example 6). Example has either 11 regions, or 8 regions (4/8 connectivity).

So how do we begin? Need to first find a pixel that contains valid value indicating a "region". Algorithm discussed: iterate for a nonzero, track current number. if connected, reuse number otherwise increment. If connectivity with multiple numbers, will flag equivalence. ultimately will perform a merge. **READ**: 2.4-2.5

4 Thursday 01/29/2026- Connectivity

Last time, searching regions. Now, *affine transforms*. Waht are affine transforms? Linear transformations are simple shift and/or scale.

- identity: $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$
- scale/reflection: $\begin{pmatrix} c_x & 0 & 0 \\ 0 & c_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$

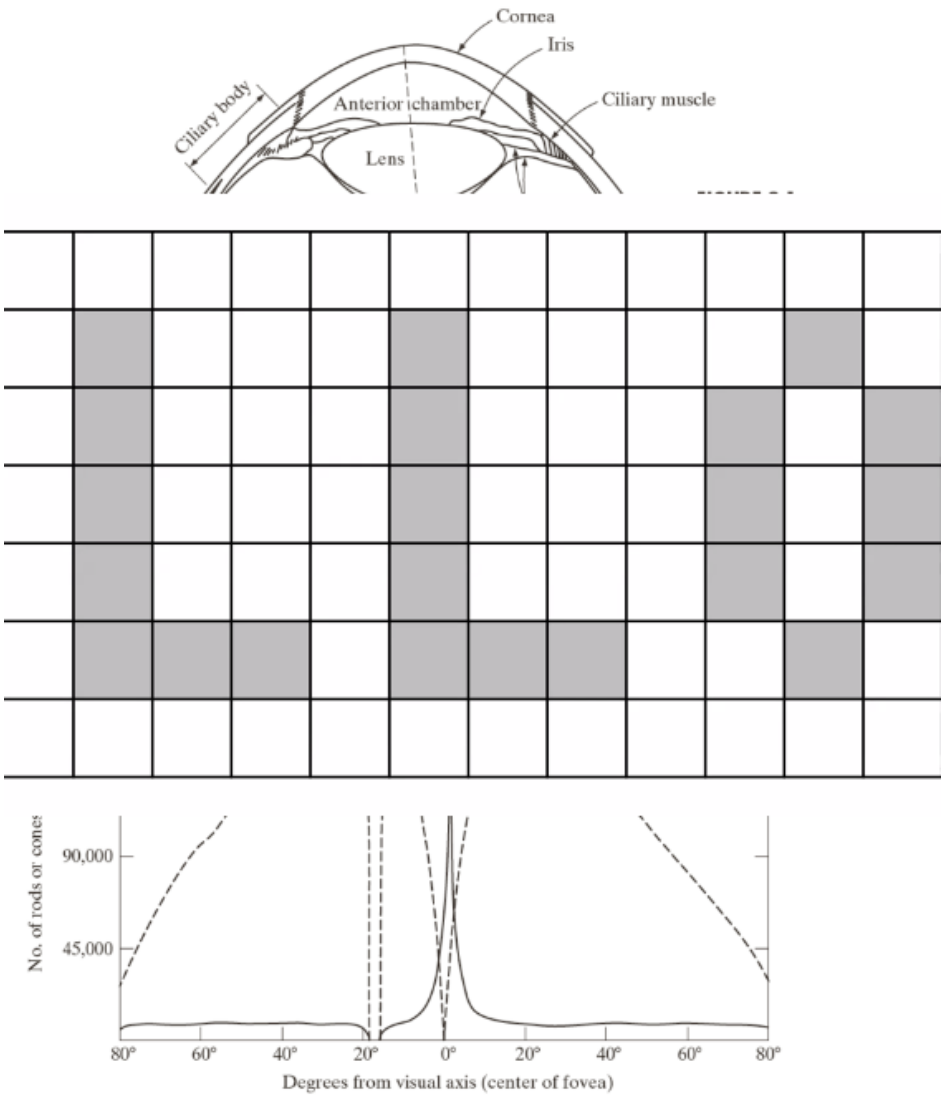


Figure 5: Illusions

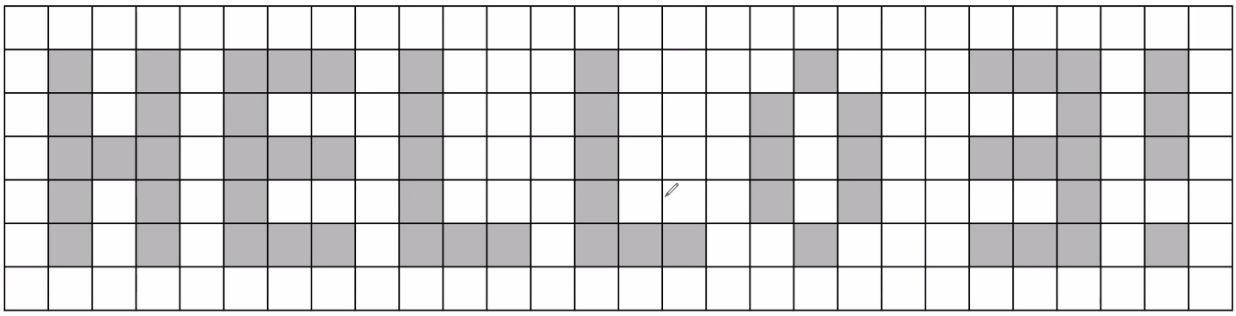


Figure 6: Illusions

- rotation: $\begin{pmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$

- translation: $\begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}$
- shear(vertical/horizontal): $\begin{pmatrix} 1 & s_w & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 \\ s_k & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

Affine is the mathematically precise. An equivalent is IEEE floating point. the exponent is the shift, so affine and nonlinear. Sounds complicated and requires a lot of work, but this makes the common case fast; dont often add exponents but only do exponent comparisons. Summary: faster to compare exponents, much more commonly used.

So what are we technically doing with these matrices? These are simply adjusting the *locations* of the values. However, our pixels in most cases don't align, so we then have to do some type of interpolation. Different types of interpolations; overall, prefer odd functions (ie linear, cubic).
So now *intensity transforms*. we're only concerned with the pixel itself and not any neighbors. Some examples include Gamma correction, negative transform, log/inverse log transform. Two things we discuss:

- monotonic transform;
- piecewise transform;

can combine PDF/CDF transformations, perform histogram transformations.
For inversion, typically it's: $max - curr$. Identifying the range is typically the challenge.

READ: 2.6

5 Tuesday 02/03/2026- Histograms

Last time, connectivity. This time, another spatial technique. We've typically done things that are more locally (ie connectivity, gamma correction). We will now look at a more distributed spatial metric. Motivation comes from gamma correction, related to monotonic functions. Let's think about probabilities; what's prob for a specific gray-scale. "Is it more centered around particular value?" "How likely is something to occur?"
In general, do you think all gray-scales are equally likely? Of course not; some pictures may be more illuminated on bright days. There maybe a particular range that's more important. So, if we know the prob distribution and we can correct for it. Prob curves are not monotonic. However, the CDF (instead of PDF) is monotonic (since it's an integral). A cumulative function is monotonic, so we can potentially use this to correct an image; what would we do with that? Taking a step back, why do we like monotonic functions? The correction using CDF helps us get an evenly balanced amount of every intensity.

So now the question is how can can we produce the CDF of an image? Sample + forming histograms. How you sample image can vary as well as how you form the histogram. A straightforward method:

- Form Histogram from input image
- Form CDF by taking integral over Histogram
- Normalize CDF
- Use normalized CDF to transform image

So another question, how can I find the background? Is it possible to find a background using CDF? Different objects may have different distributions. What happens if you don't know anything; just have to make general comparisons.
Now a question of how to compare distributions; can compare different statistical properties (ie mean, std dev, skew, etc). Allows idenfitication of unique local distributions (when compared to other locals, or general distribution).
Other ways are heuristics- if we know an object of interest has a specific characteristic, we can use that.
Overall, most the methods have not changed for the longest time. The most significant difference is the heuristics that are used and assumed.

READ: 3.1-3.3

6 Thursday 02/05/2026- Convolution

So how do we apply filters? Often times, we'll apply them in fourier domain. We can also apply them directly in the spatial domain. We're essentially doing *inner products* between image and filter (one function with another function). So why convolution? in DSP, it quantifies how a system acts when an input is put into a system. *Convolution is a type of multiplication.* An example of this is simple polynomial multiplication. If we can do things in spatial, why do we do in fourier domain? Fourier is just simple multiplication instead of convolution; in reality, the transformation from spatial to fourier is also complex so overall is still the same complexity (but using FFT improves complexity, which once again makes fourier analytics better). Good examples for images to examine: text + person, good mix of high and low frequency in an image. Note, typically when we apply convolutions, we need to apply a flip. However, most times, we have symmetric filters. Therefore, we can technically just apply correlation. Note (tangent from hw), convolution is a linear operation, so the order of applying filters via convolution is interchangeable.

READ: 3.4-3.6

7 Tuesday 02/10/2026- Fourier Transform

Today, review of Fourier Transform. What are some reasons we want to deal with the fourier domain?

- convolution is multiplication in F.domain
- for 2D, physical features all have a 'frequency' too

by definition, FT is:

$$\int f(t)e^{-j\omega t}dt$$

What we care about is the fact that if:

$$f(t) \leftrightarrow F(\omega), f'(t) \leftrightarrow j\omega F(\omega)$$

We like the exponential function because of fixed point theory, its derivative is a function of itself. We like derivatives more than integrals for similar reason: in fourier domain, we're doing multiplication instead of division. Not all integrals solvable. Similarly, can't always divide by ω unless satisfies *L'Hopitals rule*.
Some simple functions:

Definition 2. Delta function: for continuous $\delta(t)$ and $\delta[n]$; both integrals of 1, but defined differently.

integral of delta function is step function; derivative of delta function is general moment function. Integral of step function gives ramp function.
Problem with these general functions is that they have discontinuities, so technically have no rieman integrals (explainable using lebesgue theory). Note that the fourier transform is essentially an inner product; identifying similarity between a function $f(t)$ and the exponential function $e^{-j\omega t}$.
Sifting property of $\delta(t)$; if we evaluate FT on delta function, equivalent to evaluating at $t - t_0$ if taking FT of $\delta(t-t_0)$. Another interpretation is that this exponential is a rotation, or phase change (due to Euler's property). This also gives us the sampling theorem if we use multiple consecutive delta functions (comb function).

$$f(t) \leftrightarrow F(\omega), g(t) \leftrightarrow G(\omega)$$

then:

$$f(t) * g(t) \leftrightarrow F(\omega)G(\omega)$$

Can use shifting property in fourier space to identify where object properties are located and where they've been shfited to. Remember that we care about linearity, need to simply check for scale and additive properties. Higher dimension fourier transforms are simply applying 1D FT one direction at a time (this is due to the seperability of the exponential function).
READ: 4.1-4.4

8 Thursday 02/12/2026- Fourier2D (primarily matlab session)

Last time, fourier transform and properties. Today, will examine 2DFT and what you can tell about the image, particularly basic shapes. Will use function in 7 to showcase. Some comments

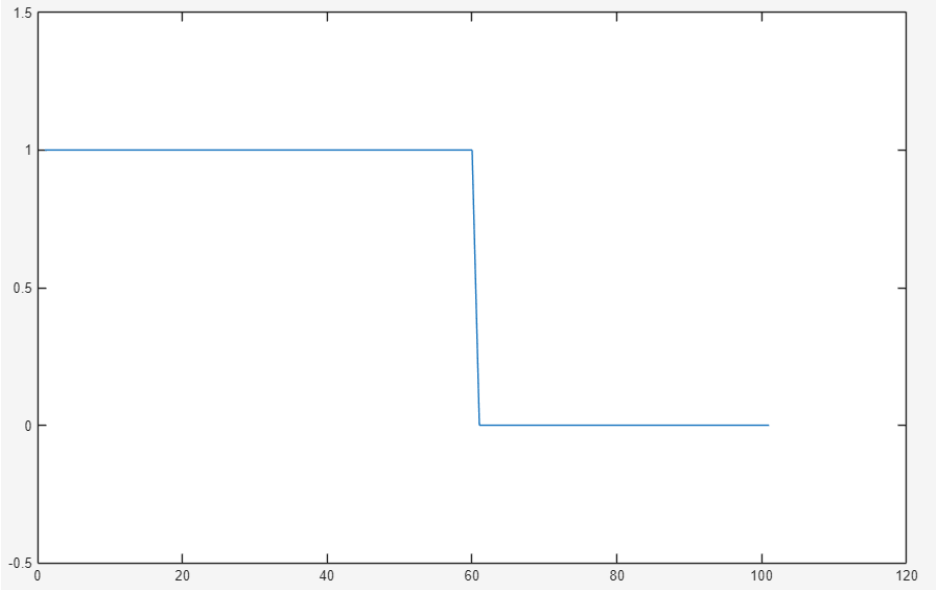


Figure 7: FFT Example- Reverse Step

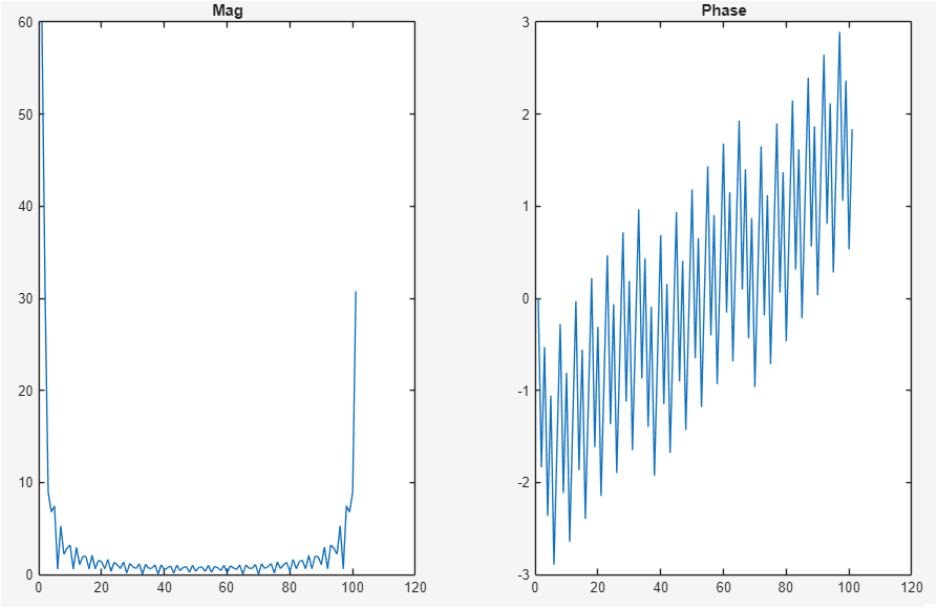


Figure 8: FFT Example- Magnitude and Phase Shift

on the plots:

- No symmetry; this is because of the need for integers, so because we have an even number

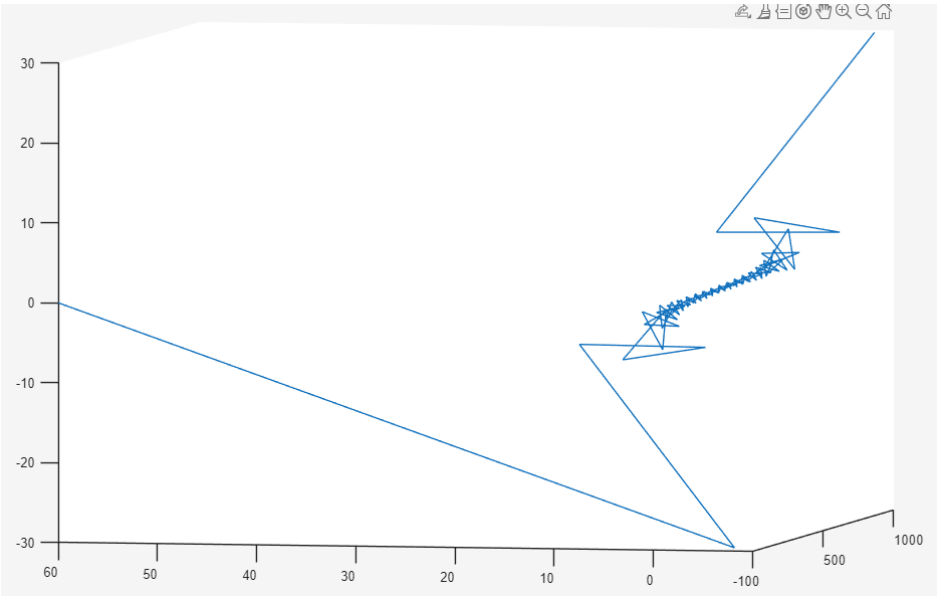


Figure 9: FFT Example- Magnitude v Phase Shift

but need to have it centered at zero, only left side is truly the freq of zero.

- The phase shows the offset.
- if we plot the magnitude vs phase shift in 3D, we can see it does a sort of a (tight) spiral.
- Typically accustomed to magnitude being centered around 0/the center. can do this using simple fftshift

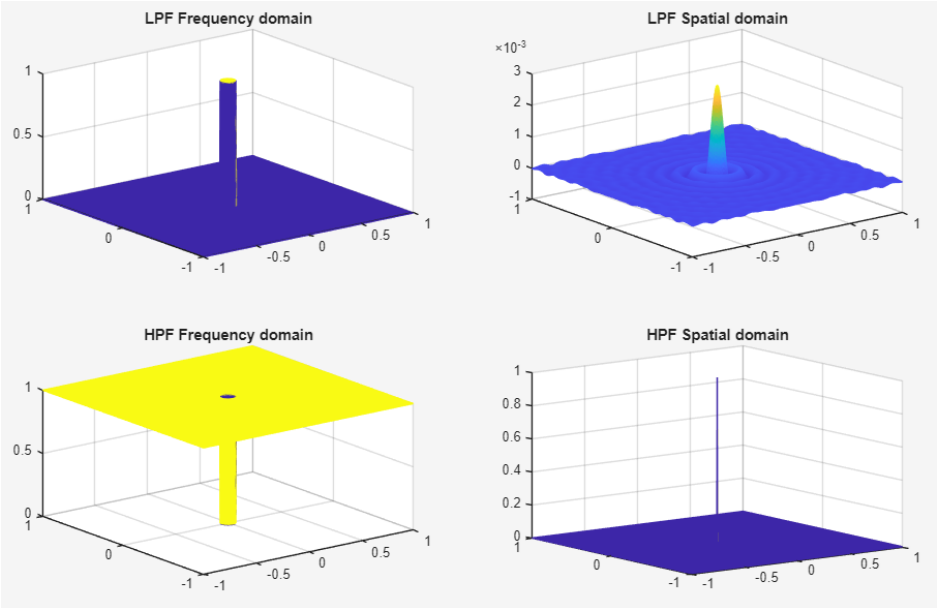


Figure 10: FFT Example- Ideal Highpass/Lowpass filters

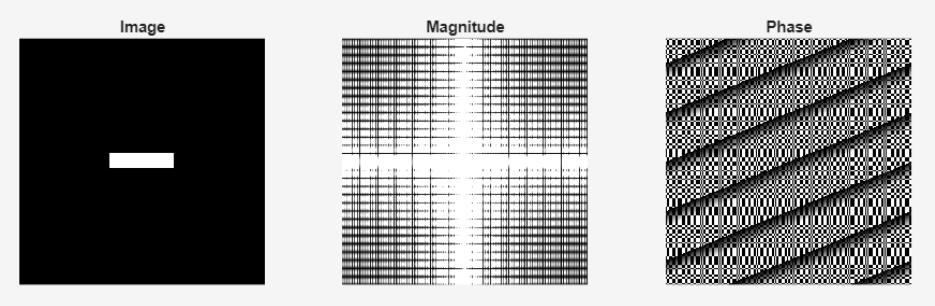


Figure 11: FFT Example- Sample Image

Figure 10 shows an example of low-pass and high-pass filters used. In our example, we use three of these at three different sizes. We will apply these to our test img in fig 11. Note that we're only getting this with ideal filters. in reality, we don't actually want ideal filters because we don't want these ripples in the spatial domain. The discontinuities are creating these ripples. In machine learning, used to also use step functions and heavy-side step functions for activation functions. For the same reason, they moved to ReLU and sigmoid instead due to smooth functions, easily differentiable. Figure 12 shows the result after applying filter. Note that we're getting these additional dark areas, more coming out with higher freq of low-pass filter. This is because of the sinusoidal waves. We have a similar effect in the high-passfilter. *A common solution to these additional effects is applying windows.* A window function allows smoother edges.Overall, changing freq will result in new shapes, new patterns based on the harmonics. In our example, we picked frequencies to perfectly line up with the image's boundaries. Overall, what's more important to an image,

Figure 12: FFT Example- Sample Image After filtering

magnitude or phase? Is one more important to another? figure 13 we combine the magnitude and phase of two different images (rectangle and palace of king david). (bottom left) introducing the phase of the palace to the magnitude of hte rect seems to produce the overall structure of the image.

The phase carries a lot of information because it tells us where everything is.

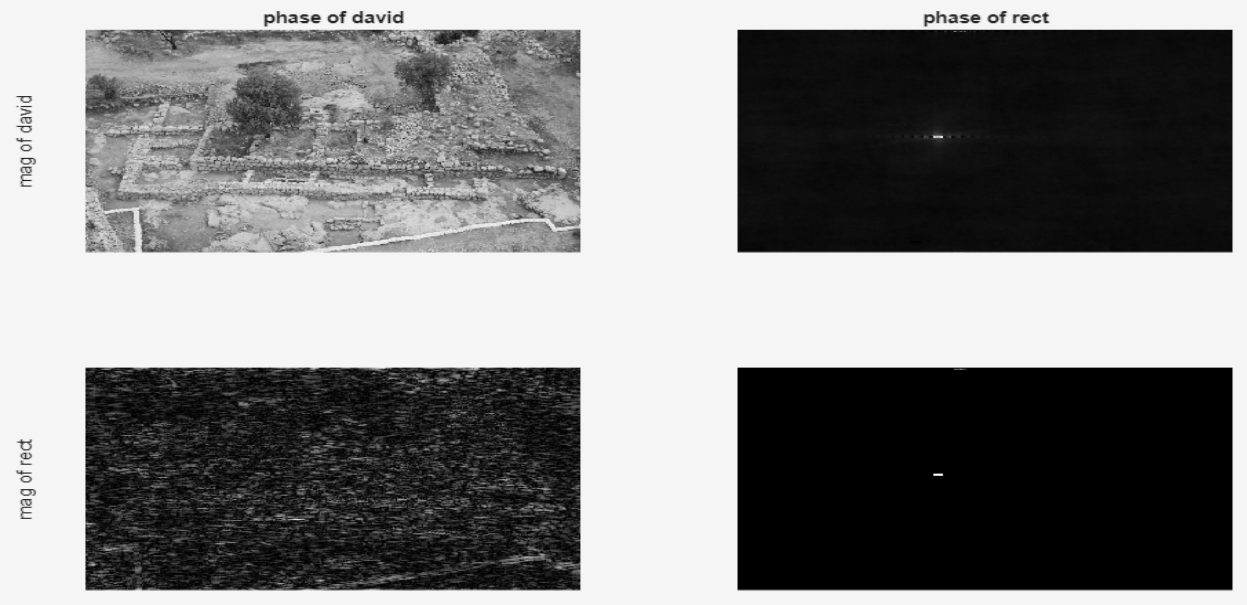


Figure 13: FFT Example- Mag v Phase Importance

READ: 4.5-4.6

9 Tuesday 02/17/2026-Fourier2D Filters(matlab)

Last time, intuitions of Fourier. So what's the problem with our ideal filters? Our ideal filters have sharp cutoffs; *finite support*. Heisenberg uncertainty principle plays a factor in this as well; if we want a perfectly sharp edge, get an infinitely large value in the freqeuncy domain. Our "ideal" is only in the frequency domain, so it's not precise in the spatial domain. These other filters provide good tradeoffs in both domains, so we pick filters depending on what we're looking for. Begin by constructing butterworth filter. In general, windowing functions introduce smoothness into filters.

```
function Ib=BLPF(I, n, D0, varargin)
    % Impelmentation of Butterworth Lowpass Filter
    % I :  image to apply filter to
    % n :  order of filter
    % D0:  cutoff frequency normalized
    % (optional) plt: plot filter

    % 1) parameters for filter
    declaredparams=3;
    [rows,cols] = size(I);
```

```
    % 2) pad to remove aliasing; pad to increase freq range
    %      tries to avoid aliasing this way
    rows2 = 2*rows;
    cols = 2*cols;
    I2 = zeros(rows2, cols2);
    I2(1:rows,1:cols) = I;
    I2F = fft2(I2);

    % 3) create filter
    H = zeros(rows2,cols2);
    A = sqrt(rows^2 + cols^2);
    for row = 1:rows2
        for col= 1:cols2
            d2 = sqrt(((row-rows)/rows)^2 + ((col-cols)/cols)^2); %note this is normalized
            H(row,col) = 1/(1+(d2/D0)^2); % theory is: F(w) = H(w)X(w)
        end
    end

    % 4) optional plotting
    if nargin>declaredparams
        if strcmp(vargin(1),'plt')
            fig = get(groot, "CurrentFigure");
            figure;
            plot((-rows:rows-1), H(:,cols));
            if ~isempty(fig)
                set(groot,'CurrentFigure',fig);
            end
        end
    end

    % 5) apply filter
    H=ifftshift(H); % this is only for matlab
    Ib2 = real(ifft2(I2F.*H)); % note numerical errors can introduce imaginary parts

    % 6) clean and return answer
    Ib = Ib2(1:rows,1:cols);
end
```

We can adjust our order to make it more or less square. If we want to convert into a high-pass filter, couple ways (that are equal):

- $H(row, col) = 1 - 1/(1 + (d2/D0)^2)$;
- $H(row, col) = 1/(1 + (D0/d2)^2)$;

Similarly, can make minor changes to create a Gaussian filter (simply use gaussian equation, mean $\mu = 0$ standard deviation $\sigma = 1$).

$$H(row, col) = exp(-1/2 * (d2/D0)^2);$$

Comparing the two, we note that Gaussian filter result isn't nearly as strong as its butterworth filter equivalent.
READ: 4.7-4.11

10 Thursday 02/19/2026-Fourier Registration

From paper we read (paper here), **cross-power** defined in it between the FT of two images. A downside is that while it accentuates differences, it also accentuates noise. Paper's idea is to match images using fourier transform; possible due to transform property of (2D) Fourier transform. Discusses translation, rotation, and scale. Translation and rotation, method works fine but not as good with scaling. In transform theory, we can use exponential and log functions because they interchange multiplication and addition; in logarthmic domain, we're not doing addition. For rotation, we can also use polar and euler's to do something similar.

In Figure 14, we have an example of if we were to shift an image in two different directions and

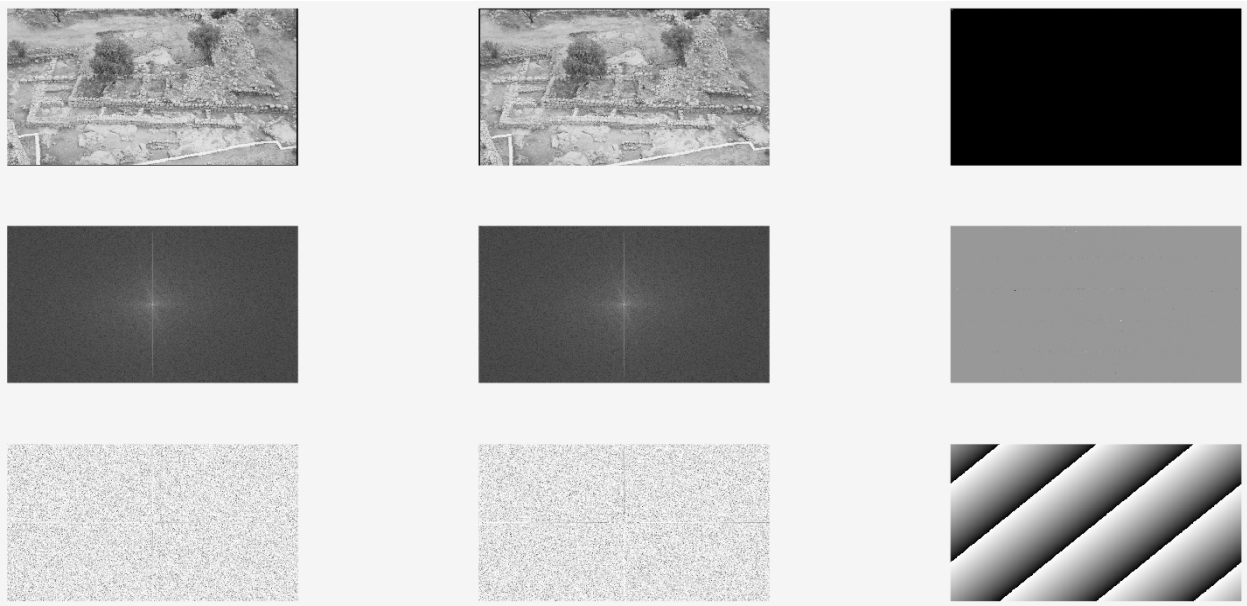


Figure 14: FFT-Shift Example

visualize their freq domains. It's not seen at this zoom but on the bottom row, can see a horizontal and vertical line. These indicate that there are shifts in both vertical and horizontal locations.

Figure 15 is a zoomed in version of grid 3 and the bottom right. Grid three, we can see there's one pixel and it's location is our shift amount. For the right, we can see the bands actually correspond to the amount of shifts too. These are just ramps. On the y-axis, the amount of drops is 2 (corresponding to shift of 2), on the horizontal, we get 3 drops (corresponding to shift of 3). So

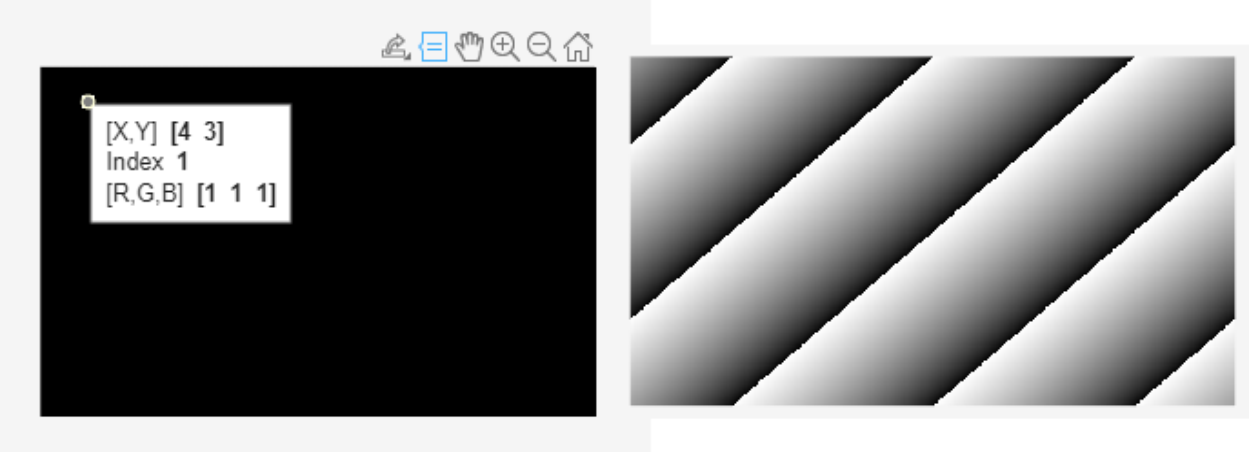


Figure 15: FFT-Shift Example

we know there's a way to identify this shift; we now want to write a function to find row/column shift. Will use shift property of FFT:

$$F_2(\xi, \nu) = e^{-j2\pi(\xi x_0 + \nu y_0)} F_1(\xi, \nu)$$

```
function [r,c] = findfftxy(ImgLook4, ImgLookIn)
    %
    % make images same size
    ImgLook4 = reshape(ImgLook4, size(ImgLookIn));
    % calcualte FT for images
    F1 = fft2(ImgLook4); F2 = fft2(ImgLookIn);
    % calculate shift exponential: F2/F1
    expShift = F2./F1;
    % convert back to spatial domain (ifft)
    impRespSpatial = ifft(expShift);
    % find biggest component and return location
    [r,c] = imMax(impRespSpatial);
end
```

Note that this is simply finding impulse resposne of a system that shifts an image!
READ: (paper)

11 Tuesday 02/24/2026

READ: (paper)
Today, how to deal with rotations and scaling. For rotations, will use polar. This will involve putting r on one axis, θ on another axis. In-fact, we will be using log-polars. This allows addition when dealing with the scaling.

First, discussing turing grayscale to log-polar. In matlab, if topleft is our origin, we only have 1 quadrant so only have rotation for 90 degrees. However, we can also shift our origin so it's in the center. If we have a total of RT rows and CT cols, center is now $((1+RT)/2, (1+CT)/2)$. Unfortunately, because we will never have exact 1-to-1 of pixels in spatial to radians, we will have to interpolate.

12 Thursday 02/26/2026- Noise

READ: 5.1-5.3
Will primarily use MatLab's random function, part of the statistics package. This function can produce values from different random distributions.
In general, we have different probability distributions because physical properties result in different random variables.
Let's begin with two uniform distributions and add them:

```
n=30000;
m=50;

x=rand(n,1);
y=rand(n,1);
total=x+y;
```

This new distribution "total" is no longer a uniform distribution! Think of the sum of two dice. When loking at the sum of two dice, it's no longer a uniform distribution (ie only 1 way for sum of 2, 2 ways for sum of 3, etc). We get a linearly increasing then decreasing distribution with a maximum at 7. So our distribution "total" becomes a triangle. If we are to instead add more and more uniform distirbutions, this results in more of a bell curve: this is an application of the *central limit theorem*! This is why we like Gaussian distributions. When we add random noise together, regardless of the distribution (with some exceptions, requires defined mean and variance), we end up with a Gaussian distribution.

```
n=30000;
t=0:20/(n-1):20;
edges=0:1/19:1;
edges(20)=1.01;
xn=random('Normal',.5,.1,n,1);
Hxn=histcounts(xn,edges);
xr=random('Rayleigh',0.05,n,1);
Hxr=histcounts(xr,edges);
xg=random('Gamma',.5,.05,n,1);
Hxg=histcounts(xg,edges);
xe=random('Exponential',0.05, n,1);
Hxe=histcounts(xe,edges);
xu=random('Uniform',0, 1, n,1);
```

```
Hxu=histcounts(xu,edges);
xsp=random('Discrete Uniform',2,n,1)-1;
temp=xu>0.95;
xsp=((xsp-.5).*temp*2+1)/2;
Hxsp=histcounts(xsp,edges);
```

- **Rayleigh**: only positive values. Common use is distribution of path lengths.
- **Gamma** and **exponential**: (gamma is general exponential) exponential is memoryless. Chance of failure at any given point. Distance between any two poisson events.
- **Uniform** same between a certain range
- **Discrete uniform** ex is dice dot totals

An additional noise type is *salt and pepper*: this is an introduction of fully saturated pixels or pixels with no saturation. This happens in our systems because it's easy to saturate while also not getting enough light. It is possible that sometimes, you only get one or the other. For example, in medical imaging, often will only get oversaturation. When we have noise, have different filters that we can apply to remove:

- different types of means (harmonic avg, geo avg, arith avg, etc)
- median
- adaptive

13 Tuesday 03/03/2026- Wiener Filter

READ: 5.4-5.8
Norbert Wiener (pronounced with v) one of greatest intellects. Discussing the Weiner filter. Assume input x going into LTI system h, output corrupted by ν and output is y:

$$y(t) = x(t) * h(t) + \nu$$

Assume $x(t)$ can be described by some statisitcal process, and assume $x(t)$ independent of ν ($x(t)$ and ν orthogonal, or integral of product is 0). For the Weiner filter, takes in as inputs: shift H, noise ν. It's assumed you know what these are. It's a challenging figuring out these things before you can even apply the filter. First, determine power within an image. Our correction is therefore:

For the Weiner filter, takes in as inputs: shift H, noise ν. It's assumed you know what these are. It's a challenging figuring out these things before you can even apply the filter. First, determine power within an image. Our correction is therefore:

```
Hstar=conj(H);
Hw=Hstar./(max(abs(H).^2+(noise^2)./Pim,delta));
I=mat2gray(min(max(real(ifft2(Hw.*FI)),0),1));
```

The Hw can be considered SNR. Incorporated is a check for division-by-zero as well as a $[0, 1]$ bound.
Side Note: It's true that in our case, we are are technically know exactly the system how we formed our noisy image so know how to deal with it. However, we can always test with an impuse response, or a delta function equivalent, in order to find $H(\omega)$. This is also known as "calibration".

14 Thursday 03/05/2026-(more) Filtering

READ: 5.9
Today, dealing with the idea of *constrained filtering*. In general, three:

- **Smoothing**: generally means you samples up to a certain point, but want to determine a value in the past (and you have points around what we want); this is interpolating
- **Filtering**: generally for our current point
- **Prediction**: this is extrapolating

Laplacian of an image is good for identifying sharp discontinuities. Compared to Weiner filter which assumed we knew the noise spectral density, here we can make tweaks. Going back to basic calculus, we used derivatives. Remember, local mins were found when the first derivative was zero and second derivative is positive. In general, min/max either requirements of derivatives, OR extreme points.

15 Tuesday 03/17/2026-Filtered Backprojection

READ: 5.11
Last time, constrained least squares. Many versions to consider. In implementation, involves laplacian filter. There are also different types of laplacian filters. Numerically, there are different ways to compute Laplacian; this is why there are different Laplacian filters that can be used.

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix} \tag{1}$$

In our loop, we don't know what the truth is. So in order to check if our noise system estimate is correct, we then have to reapply the noise to our denoised image and take the difference, this is called *residual*:

Definition 3. If system is $F()$, measured image with noise is I and denoised is I', the **residual** r is:

$$r = F(I') - I$$

can also take the norm of the residual to quantify it.

Iteratively, if our norm is too big or too small, we adjust our γ.

```
function I = constrained_filter_starter(Ib,H,m,v,a,g)
     % m = mean, v = variance
     [rows,cols]=size(Ib);
     fIb=fft2(Ib-m);

     Hstar=conj(H);
     H2=abs(H).^2;
     %Laplacian measures the second derivative
     %fft of Lap with padding
     Laplacian = [0 -1 0; -1 4 -1; 0 -1 0];
     P=fft2(Laplacian,rows,cols);
     P2=abs(P).^2;

     % estimated noise contributions
     eta2=rows*cols*v;

     %Starting value of gamma
     gamma=g;
     % how to change gamma
     gamma_frac=0.9;
     %iteration setup
     condition=1;
     steps=0;
     max_steps=50;
     while condition
          %Calculate and apply filter
          G=Hstar./(H2+gamma*P2);
          fI=G.*fIb;
          %check quality by calculating the residual
          r = Ib - real(ifft2(H .* fI));
          r2=norm(r,'fro')^2;
          %give feedback on how we are doing
          disp(['noise = ',num2str(eta2),'   residual = ',num2str(r2),'   gamma = ',num2str(ga
          %adjust gamma
          if r2<eta2-a
               gamma=gamma*gamma_frac;
          elseif r2>eta2+a
               gamma=gamma/gamma_frac;
          else
               condition=0;
          end
          steps=steps+1;
          if steps>max_steps
               condition=0;
```

```
        end
    end
    fprintf('Exiting with %i iterations\n', steps);
    % image desired
    I=ifft2(fI);
end
```

Now going into *filtered backprojection*. typically in detectors, it's only one energy level. Can also use dual energy to identify additional interior materials. For scanning, It's often better to offset slightly (implementation of xray on scanning an object). Additionally, detectors being offset improves sampling for "conjugate rays".

16 Thursday 03/19/2026-FBP

READ: 5.11
Continuing more into FBP code.
First must identify size of the image to be reconsturcted:

- rotation produces circle
- diam of circle is simply width of projectors
- reconstructed image is square that fits within the circle
- an extra, we impose demand that our recon img is even

In general for ieee floating point, using "round" actually alternates between up and down for 0.5; this is to avoid an upward bias.
Now, the general idea is that we're going to take our projections, build it up in teh 2D fourier space, then iFFT to get our reconstruction. However, we can speed this up by applying FFT on 1D. This is possible due to the **central slice theorem**. In general, we end up with more low frequencies because the evenly spaced detectors result in evenly spaced sampling, but in the polar grid this is more densely sampled near the origin. this is another visualization of how/why the ramp filter is used in FBP.
Consturcting ramp filter:

- we want a 0 to correspond with 0 in freq; this means we're cancelling out all constants
- if even length, we will have 1 maximum value ; 2 (equal) otherwise

We then want to determine the coordinates corresponding to the center of the detector.

17 Tuesday 03/24/2026-FBP

READ: 9.1-9.4
(walking through FBP code)
After constructing the ramp filter, need to apply this. Iterate through the sinogram for each angle, FFT and apply the filter, then iFFT. We then smear/backproject this result across the imagine the desired angle. Remember, we need only do 1D FFT because of the central slice theorem. In implementing the smear backwards, need to select a way to define a line.

18 Thursday 03/26/2026

READ: 9.5
New area. Just finished up on filtered backprojection (chapter 5). Now discussing morphology. Simple yet extremely powerful idea. What we're going to be doing is applying a structural element. For now, begin with boolean then extend to grayscale and rgb. Overall, if our pixel has an "on", we will then either dilate or erode.

- if "on", dilate (or expand)
- if "off", erode (or remove)

Applications include filtering, identifying shapes, gray-scale correction.
Discussion of some matlab builtin functions:

- strel('disk',3,8) "structure element", creates disk
- imfilter()
- Erode: removed text

Combining:

- Dilation - Img This gives us an (outer) edge detection. Taking laplacian and subtracting will give similar but not exact; for dilation, it only goes outwards (whereas laplacian will go both directions).
- Img - Erode This provides us with the interior of the different edges. Great way to isolate texts without any shadow effect.
- Open (Erode $\rightarrow$ Dilate) Removes small light areas
- Close (Dilate $\rightarrow$ Erode) Removes small dark areas

Note that these are not invertible operations. You can perform successive operations and still obtain changes. Open/close great for removing salt+pepepr noise.

19 Tuesday 03/31/2026- Using Morphology for Boundary and Holes

READ: 9.6
Morphology: Dealing with sets of points, typically 0 and 1. How they relate to eachother defined by a structure element, the "morpho" or "form". Two most common operations are eroding and dilating (removing boundary elements or adding new ones in). From there, we combined to introduced two more operations: open and close.
Today, will deal with rgb images.

- Img - erode(Img) = edge detection
- Dilate - Erode = outer boundary

20 Thursday 04/02/2026

READ: 10.2

21 Tuesday 04/07/2026 (OUT SICK)

READ: 10.3

22 Thursday 04/09/2026

READ: 10.4
Discussing canny edge detector code...
Otsu's method: provides thresholding (to perform segmenting). "If there are two objects, there should be two histograms. Even with histograms, we should find two different groups; look at different groups to try splitting. Want to maximize cross-entropy".

23 Thursday 04/16/2026-Region Growing

READ: 10.5
How do you characterize a region? Can look at spatial regions or perhaps sizing of objects. Want to define some starting point. Perhaps (light) intensity. All these different characteristics jump out.

Looking at it from a different pov, what if we use a surface mapping? how would you characterize regions? nonexhaustive list of some methods:

- mean and standard dev
- mean and threshold
- max change
- upper/lower
- cross entropy; find division point where you find the biggest change between a single pixel and its neighborhood.
- combo of the above

The general plan for region growing:

1. starting seed
2. get stats
3. find candidates
4. check if candidates meet requirement
5. if region grew, then repeat steps

How might I be able to check all the candidates nearby? Can simply walkthrough

24 Tuesday 04/21/2026-Watershed Segmentation

READ: 10.6
Watersheeding is different way of grabbing regions. originated in 1970s, but practical algorithm not until 1991. For visualization, you're filling in basins one by one; basin that is deepest gets filled first. Each basin is a region. In this problem, as two basins might begin to merge due to lack of separation, we introduce a **dam**. This is to distinguish two basins so no merging occurs.
How about if we have a basin of only 1 pixel? maybe we don't want this since it's just noice.
How would we begin this algorithm?

- lowest parts
- positive hessian + = deriv
- dynamically add isolated regions; this is a challenge with dams

Another way to fil this is "raindrops method"; two methods. Random and flowmap:

- random: add random "drops"; flow downhill until cant move anymore
- flowmap: each pixel notes where it flows to; pixels that dont flow anywhere grouped in regions.

25 Thursday 04/23/2026-Motion Segmentation

READ: 11.5
how to detect motion? frame difference. In general we won't be able to tell where or what, only that something is moving/ has moved away when pixel intensities change. Note that the range of our possible values doubles in our difference frame; if our pixels are range $[0, 1]$, then the difference has a possible range of $[-1, 1]$. If we're tracking using video, we will also need a reference frame: call this **golden reference** or **frame 0**. So how can we get these reference frames? Sometimes we can easily obtain this. Othertimes, if our frame is always busy, we will need to combine multiple frames. Another issue that occurs is differing exposure (think of sunlight, exposure). Shadows too from buildings can also produce subtle changes that are still detected.

26 Tuesday 04/28/2026-PCA

READ: 12.5-12.7
General idea of "component analysis" is to break down signal or image into basic components, ie eigenvectors. PCA specifically looks into singular values. Largest component tells you it has the biggest contribution to your image; we call this the "principal component". With principal components, we assume a gaussian system.
Why Gaussian? Gaussian through a linear system produces Gaussian output. We especially like gaussian because if we add up enough random distributions, they become a gaussian distribution. This is the **Central Limit Theorem (CLT)**.
Mean and standard deviation:

- 1 standard deviation = 68.2% of area
- 2 std dev= 95.4% of area
- 3 std dev = 99.7% of area

From a geometric view: standard deviations form ellipses. If we want to fit a line, a couple ways we can do it based on different metrics:

- minimum residual
- maximum variance

Least squares is typically just the vertical different of the points to the approximate line. Residual minimization takes into account both x and y distance. the two above are equivalent; the second

is maximizing Algebraicly, we have a set of measurements:

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix}$$

Each component represents mean and variance. We're overall solving $Ax = b$.

- variance: $\sigma_i^2 = \frac{1}{n-1} A_i^T A_i$
- covariance: $\sigma_{ij}^2 = \frac{1}{n-1} A_i^T A_j$
- combined: $C = \frac{1}{n-1} A^T A$

Note the $A^T A$: this is just the normal matrix; this is simply the weight factor we need to divide by.
Eigenvalue decomposition

$$Cx = \lambda x \tag{2}$$

This is important because it's a fixed point in our system. By multiplying by x, essentially saying we stay in the same direction. We can also stack all of these vectors x and form matrix X:

$$CX = X\Lambda \tag{3}$$

This means we can then find C as such:

- $C = X\Lambda X^-1$
- $C = X\Lambda X^T$

A better way to calculate is using singular value decomposition: If $A = U\Sigma V^T$:

$$C = \frac{1}{n-1} A^T A = \frac{1}{n-1} V\Sigma^2 V^T \tag{4}$$

So what we're going do to:

- we have indep gauss var
- ahve linear combo of them
- what we want: lowest dimensional basis, or maximum information/variance
- to solve: assume zero mean and take SVD. the right singular vectors will form a basis and right singular vectors will project to the basis (using unitary matrix). These are ordered so that first column contains most information, second has second most, etc.

For exampl, take 2 sensors, each colelcting n data points (x and y). We want to take the principal components and rotate the data:

$$A = [xy], n = 200 \tag{5}$$

Now take SVD: $U\Sigma V^T = \text{svd}(A)$, for example:

$$V = \begin{pmatrix} 0.8634 & -0.5044 \\ 0.5044 & 0.8634 \end{pmatrix}$$

Here, $\theta \approxeq 30.29\,\text{deg}$. So given A, we can have $Av = U\Sigma$. This will rotate all our points so they're not along the x-axis or "projecting on the the plane" and now our primary component is in the x axis. One great application is in visualization: for example if we many components, we can simply display the main components

27 Thursday 04/30/2026

28 Extra

Growth: Acknowledge the error, what went wrong, make the correction- KES