1 Introduction

✠ A pixel is not a square (or rectangular). It is simply a point sample. There are cases where the contributions to a pixel can be modeled, in a low order way, by a little square, but not ever the pixel itself. The sampling theorem tells us that we can reconstruct a continuous entity from such a discrete entity using an appropriate reconstruction filter - for example a truncated Gaussian.

✠ Cameras approximated by pinhole cameras. When pinhole too big - many directions averaged to blur the image. When pinhole too small - diffraction effects blur the image.

✠ A **lens system** is there in a camera to correct for the many **optical aberrations** - they are basically aberrations when light from one point of an object does not converge into a single point after passing through the lens system. *Optical aberrations* fall into 2 categories: monochromatic (caused by the geometry of the lens and occur both when light is reflected and when it is refracted - like pincushion etc.); and chromatic (Chromatic aberrations are caused by dispersion, the variation of a lens's refractive index with wavelength). Lenses are essentially help remove the shortfalls of a pinhole camera i.e. how to admit more light and increase the resolution of the imaging system at the same time. With a simple lens, much more light can be bought into sharp focus.

Different problems with lens systems:

1. **Spherical aberration** - A perfect lens focuses all incoming rays to a point on the optic axis. A real lens with spherical surfaces suffers from spherical aberration: it focuses rays more tightly if they enter it far from the optic axis than if they enter closer to the axis. It therefore does not produce a perfect focal point and results in image blurring.

2. **Vignetting** - There are several causes of vignetting and one of them is optical vignetting which is caused by the physical dimensions of a multiple element lens. Rear elements are shaded by elements in front of them, which reduces the effective lens opening for off-axis incident light. The result is a gradual decrease in light intensity towards the image periphery.

3. **Scattering** at the lens surface - Some light entering the lens system is reflected off each surface it encounters.

4. **Barrel** and **Pincushion** distortion.

✠ Different kind of optical sensors include **Charge coupled devices (CCD)**. It is simply an array of **photosites** which can be thought of as buckets to hold electrical charge. The amount of charge stored is proportional to the incident light intensity during exposure. The digitization is done using ADCs line by line. Since the a photosite can get saturated, it can cause **blooming** in the final image. Also a CCD can produce a thermally generated image (in darkness) - which is called the **dark current**.

✠ Different samplings:

1. **Undersampling** - is the process of having pixels sampling below the *Nyquist rate* (which is twice the frequency of the original signal). *Undersampling* causes **aliasing** which makes different signals become indistinguishable when sampled.

2. **Oversampling** - is the opposite of *undersampling* i.e. sampling higher than the *Nyquist rate*. The downside of *oversampling* is the waste of storage resources. Still taking more samples with the same number of photons per pixel improves *Signal to Noise ratio* (SNR).

✠ Signal to noise ratio in imaging is usually measured as:

$$\text{SNR} = \frac{\mu}{\sigma}, \qquad \sigma = \sqrt{E[(X-\mu)^2]}, \quad \mu = E[X]$$

Although the industry standard is to take $20\log_{10}(\mu/\sigma)$ for the SNR. If give multiple images of the same scene, the same test can be used on a per-pixel basis. The quantity $\sigma(i,j)$ would give an estimate of the standard-deviation of the acquisition noise at that pixel. The average of $\sigma(i,j)$ would give an estimate of the average noise.

The **auto-covariance** of the image shows cross-talking of noise between adjacent pixel. It is computed as the covariance of the signal against pixels in its neighborhood $E[(X_{ij}-\mu_{ij})(X_{i'j'}-\mu_{i'j'})]$.

✠ The **Nyquist Frequency**, is half the sampling frequency of a discrete signal processing system (not to be confused with *Nyquist rate* which is used for a continuous time signal). The sampling theorem shows that aliasing can be avoided if the Nyquist frequency is greater than the maximum component frequency of the signal being sampled.

Quantization converts a real valued function to integer / digital / discrete values. After quantization the original signal cannot be reconstructed, whereas in sampling, which is above the *Nyquist frequency*, the signal can be reconstructed.

✠ A **Bayer pattern** is used to for arranging RGB color filters on a square grid of *photosites*. It uses twice as many green elements as red or blue to mimic the physiology of the human eye. The retina has more rod cells than cone cells and rod cells are most sensitive to green light.

✠

1. **Image resolution** - is the number of pixels to represent an image.

2. **Geometric resolution** - is the number of pixels per area.

3. **Radiometric resolution** - is the number bits per pixel.

✠ Usually the temporal resolution of the eye is 10Hz for rods and 60-100Hz for cones. Eyes also have good quantization resolution - it can discern objects with a dynamic range of 90dB. Although it cannot achieve such a dynamic range at the same time (which is equivalent to saying that humans cannot hear a whisper in a loud environment - but cannot perceive both levels of sound in isolation). At a particular moment an eye can usually perceive a contrast ratio over a range of 26dB to 40dB.

2 Segmentation

✠ **Segmentation** partitions an image I into N regions:

$$I = \bigcup_{i=1}^{N} R_i, \quad R_i \cap R_j = \emptyset, \ \forall i \neq j$$

✠ Types of segmentation:

1. *Binary* vs. *Multi-region*
2. *Supervised* vs. *Unsupervised* - supervised is where a human annotator can help along the way.
3. *Exact* vs. *Over-segmentation* - Over-segmentation aims to just divide image into segments, where each segment belongs to a single surface in the real world - although that surface might be broken into multiple segments.
4. *Bottom-up* vs. *Top-down* - Bottom-up works on visual coherency, whereas top-down works on semantic coherency.
5. *Pixel-based* vs. *Boundary-based* - Boundary-based algorithms include snakes and other contour approaches.

✠ **Thresholding** is a binary segmentation process where each pixel is marked as *in* or *out* of the region of interest:

$$B(x,y) = \begin{cases} 1 & \text{if } I(x,y) \geq T \\ 0 & \text{if } I(x,y) < T \end{cases}$$

✠ Given ground-truth data, the **Receiver Operator Characteristic (ROC)** is a graph of true positives/P against false positives/N. By setting different thresholds (or any other characteristic), we can get different set of performance values. This allows us to build a whole curve. The more the curve is towards the top left corner, the more perfect the performance of the algorithm is. The area under the curve $A \in [0, 1]$ is a good metric for judging the performance.

Remember:
FP + TN = N, #total number of negatives;
TP + FN = P, #total number of positives.

✠ To choose the best threshold, we select the point where the gradient is:

$$\beta = \frac{N(V_{TN} + C_{FP})}{P(V_{TP} + C_{FN})}$$

where V_{TN} and V_{TP} are values of the true negative and true positive respectively; and C_{FP} and C_{FN} are the costs of false positive and false negative respectively. Usually $V_{TN} = V_{TP} = 0$.

✠ **Pixel neighborhoods** are defined as 4-neighborhood or 8-neighborhood. Both give rise to pixel *connected paths* and *connected regions*. Using a binary image, we can **Connected component label** (multi-region segmentation) by assigning a unique label to each (4 or 8) connected region. Can be done recursively in code - but need to decide on a search pattern for the neighborhood pixels - it looks more like depth-first search

✠ Given a seed pixel, we can get binary segmentation by **Region growing** to connected regions from the seed pixel. It can be codeed using a queue of pixels - making it look like breadth-first search.

✠ **Watershed Segmentation** segments grayscale images and can be looked at as flooding the grayscale topology with water. Starting at the lowest pixel value, it floods basins and marks segment boundaries wherever different water catchments meet, when water level rises. This technique is often applied on the *gradient* image. Typically has the problem of over-segmentation. It works by:
(1) pushing finding local minimma pixels into a queue (ordered by grayscale values). These seed pixels may be specified.
(2) Pop pixel(s) from queue (with minimum value).
(3) Label neighboring pixels accordingly and add them to queue.
(4) Repeat from step (2) until queue empty.

✠ If we look at segmentation as a *clustering* problem, we can used *K-Means* approach - which divides pixels into a set of k clusters based on their euclidean distance:
(1) Randomly generate k mean vectors.
(2) Assign each observation to one of k clusters whose mean vector is the closest.
(3) Calculate the new means to be the centroid of the observations in the cluster.
(4) Repeat from step (2) until no change in assignment.

The approach is sensitive to outliers; plus we need to pick a k, the number of clusters. Note that it can produce disjoint segments.

✠ Using a limited sized window, we can keep shifting the mean vector by looking at the mean of points in the window. This is done until convergence. This method is **Mean Shift**. The step of moving the mean vector is repeated until convergence. This iterative process is done for each pixel; and a cluster is eventually formed of all pixels which lead to the same convergent mode. Also we can merge clusters whose mode is nearby. This technique is robust to outliers, although the output is highly dependent on the window size chosen.

Note that since both *K-Means* and *Mean-Shift* work in feature space for pixels, they can result in disjoint segments.

✠ **Efficient Graph based image segmentation** Felzenszwalb 2004:

1. Start with each pixel being a segment. From here we would merge segments.
2. The internal difference of a segment is defined as:

$$Int(C) = \max_{e=MST(C,E)w(e)}$$

   where $MST(C, E)$ denotes the minimum spanning tree of the segment C with edges E each with a weight $w(e)$. The intuition they propose for this measure is that this MST can only remain connected if edges of atleast weight $Int(C)$ are considered.
3. The difference between two segments C_1, C_2 is defined as:

$$Dif(C_1, C_2) = \min_{v_i \in C_1, v_j \in C_2, (v_i, v_j) \in E} w(v_i, v_j)$$

   i.e. the minimum weighted edge between two segments. If there is no edge $Dif(C_1, C_2) = \inf$
4. In each step we can merge segments if this criterion is met:

$$Dif(C_1, C_2) < \min(Int(C_1) + \tau(C_1),\ Int(C_2) + \tau(C_2)) \ = MInt(C_1, C_2)$$

   where $\tau(C) = k/|C|$. This dictates if the difference between the different components is smaller than any of the two component edge differences, there is little chance of a boundary between the two components, hence they can be merged. The $\tau(C)$ is there to provide some support for merging only in absence of a strong edge - it also helps grow regions from single pixels ($|C|$ denotes the size of the component, and k is a constant).
5. Each step involves sorting un-connected edge strengths. Keep repeating until we can't make any more merges.

6. Discusses different edge weighting schemes $w(v_i, v_j)$, in which it chooses a function which is based on the absolute intensity difference between pixels connected by an edge. They show results using a feature space built on (x, y, r, g, b).

3 Transformations

✠ Gray Level transformation changes gray level of each pixel by $f : \mathbb{R} \to \mathbb{R}$ such that $I_2 = f(I_1(x, y))$. There are two kinds of functions (all graphs are taken I_2 against I_1:

1. Linear function:
$$f(x) \;=\; \alpha x + \beta$$
- while preserving the range (0, 255). Positive β increases overall brightness and vice-versa. Increasing α (making graph more steep) increases scaling of gray level values.

2. Non-linear function (**Gamma Correction**)
$$f(x) \;=\; 255^{1-\gamma} x^{\gamma}$$
It helps adjust for differences between camera sensitivity and human eye. The graph bends from the constant gradient line according to γ. Smaller the γ, more bending to the I_2 side; increasing the brightness of the image. Larger the γ more bending towards I_1; decreasing the brightness of the image.

3. **Histogram equalization**: aims to make the histogram flat, or make the histogram or to give a constant gradient cumulative histogram. It is simply done by building the histogram and then assigning the histogram's values to individual pixels according to their gray scale values.

✠ **Geometric transformations** include **Affine transformations**:

$$\begin{aligned}(x',\ y') &= T(x,\ y)\\ \begin{bmatrix}x'\\ y'\end{bmatrix} &= \begin{bmatrix}a & b\\ c & d\end{bmatrix}\begin{bmatrix}x\\ y\end{bmatrix} + \begin{bmatrix}t_x\\ t_y\end{bmatrix}\end{aligned}$$

Affine transformation allow translation, scaling, rotation and shearing. The affine matrix is given in the form of rotation/scaling followed by scaling/rotation followed by translation. It is easier to use T^{-1} and then map pixels from I_{src} to I_{dst}, by iterating over the pixels of I_{dst}. Since $T^{-1}(x, y)$ does not always give us integer coordinates, we need to interpolate. **Nearest neighbor** interpolation works by picking the nearest pixel. Arguably the most famous one is **bilinear interpolation**:

$$I_2(x', y') = \Delta x \Delta y I_1(x_2, y_2) + (1 - \Delta x)\Delta y I_1(x_1, y_2) + \Delta x(1 - \Delta y) I_1(x_2, y_1) + (1 - \Delta x)(1 - \Delta y) I_1(x_1, y_1)$$

where $(x, y) = T^{-1}(x', y')$, and (x_1, x_2, y_1, y_2) are the four closest neighbors of (x, y)

Quadratic interpolation fits a bi-quadratic function on a 3×3 neighborhood of grid points; and **Cubic interpolation** fits a bi-cubic function on a 4×4 neighborhood of grid points.

✠ *Affine transformations* are 1^{st} order polynomial transformations which maps lines to lines. **Polynomial transformations** like the **Quadratic warp** can bend lines:

$$x' = a_0 + a_1 x + a_2 y + a_3 x^2 + a_4 xy + a_5 y^2$$
$$y' = b_0 + b_1 x + b_2 y + b_3 x^2 + b_4 xy + b_5 y^2$$

This can also allow us to do **control point warps**. In this the user gives m points (since this quadratic warp we need $m \geq 6$ points) and their mappings. We use them to figure out the coefficients which are used to map all the remaining points:

$$\begin{bmatrix} x'_1 & y'_1 \\ \vdots & \vdots \\ x'_m & y'_m \end{bmatrix} = \begin{bmatrix} 1 & x_1 & y_1 & x_1^2 & x_1 y_1 & y_1^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_m & y_m & x_m^2 & x_m y_m & y_m^2 \end{bmatrix} \begin{bmatrix} a_0 & b_0 \\ \vdots & \vdots \\ a_5 & b_5 \end{bmatrix}$$
$$X_2 = X_1 P \quad \Rightarrow \quad P = (X_1^{\mathrm{T}} X_1)^{-1} X_1^{\mathrm{T}} X_2$$

✠ **Image registration** involves determining the ideal transformation that aligns two similar images. This can be done by optimization techniques that maximizes the similarity between two images (like inverse of SSD):

$$S(I_1, I_2) = -\sqrt{\sum_{x \in I_1} (I_1(x) - I_2(x))^2}$$

You can also use cross-correlation or mutual information.

4 Morphological Operators

✠ **Morphological operators** are used for local pixel transformations for processing region shapes usually in binary images.

✠ Simple 8-neighbor **Erosion** (*Minkowsky subtraction*) is done by erasing any pixel set to 1, if one of its 8-connected neighbor is set to 0.

Similarly, 8-neighbor **Dilation** (*Minkowsky addition*) is done by setting any 0 pixel to 1, if one of its 8-connected neighbor is 1.

These are used for smoothing region boundaries, or for removing noise/artefacts.

✠ *Morphological operations* take two arguments: (1) the input image, (2) the binary structuring element whose center is known. The operation proceeds by aligning the structuring element (strel) over each pixel and then comparing the pixel neighborhood with the strel neighborhood to get the output.

Binary images and strel's can be looked with set notation where a pixel is included in the set of it is set to one:
(1) Union: $I_1 \cup I_2$
(2) Intersection: $I_1 \cap I_2$
(3) Complement: $I^C = \{x : x \notin I\}$
(4) Difference (relative complement): $I_1 \backslash I_2 = \{x : x \in I_1 \text{ and } x \notin I_2\}$
(5) Symmetry around point $\underline{o}$ (acts as 180°rotation): $\breve{S} = \{\underline{o} - x : x \in S\}$

✠ Fitting, Hitting and Missing, where S is the strel and I is the image:

1. S **fits** I at x if:
$$\{y : y = x + s, s \in S\} \subset I$$
2. S **hits** I at x if:
$$\{y : y = x - s, s \in S\} \cap I \neq \{\emptyset\}$$
3. S **misses** I at x if:
$$\{y : y = x - s, s \in S\} \cap I = \{\emptyset\}$$

Note $x + s$ is translation of s to x, and $x - s$ is the translation of 180° rotated s to x.

✠ Different morphological operations:

1. Erosion:
$$E(x) = \begin{cases} 1 & \text{if } S \underline{\text{fits}} \ I \text{ at } x \\ 0 & \text{otherwise} \end{cases}$$
$$E = I \ominus S = \{x = x + s \in I \text{ for every } s \in S\}$$
2. Dilation:
$$E(x) = \begin{cases} 1 & \text{if } \breve{S} \ \underline{\text{hits}} \ I \text{ at } x \\ 0 & \text{otherwise} \end{cases}$$
$$E = I \oplus S = \{x = x - s \in I \text{ for any } s \in S\}$$

Some rules to remember:

1. Erosion and dilation are dual operations:

$$(I \ominus S)^C = I^C \oplus \check{S}$$

2. Commutativity and associativity:

$$I \oplus S = S \oplus I \qquad (I \oplus S) \oplus T = I \oplus (S \oplus T)$$
$$I \ominus S \neq S \ominus I \qquad (I \ominus S) \ominus T = I \ominus (S \oplus T)$$

✠ **Opening** is *erosion* followed by *dilation*:

$$I \circ S = (I \ominus S) \oplus S$$

Closing is *dilation* followed by *erosion*:

$$I \bullet S = (I \oplus S) \ominus S$$

Both of them are idempotent operations. They are also dual operations:

$$(I \bullet S)^C = I^C \circ \check{S}$$

Opening has the property of retaining holes while remove noise (which was originally in the form of small speckles) while *closing* has the property of removing holes and expanding noise. This behavior should be evident from the ordering of *erosion* and *dilation* in both operations. To remove holes in the foreground and islands (noise) in the background, do both *opening* and *closing*. Usually *opening* followed by *closing* performs better for removing noise.

✠ **Granulometry** provides a size distribution of distinct regions or "granules" in the image (can be related to counting red-blood cells). The method is simple (1) open the image with increasing size of the strel, (2) count the number of connected component regions. We can then plot number of regions against strel radius / size. The negative derivative of this graph is called the **Granulometric Pattern spectrum**. Wherever it is close to 0 for a wide range of strel sizes, it gives a rough estimate of the number of elements in the image.

✠ The **Hit-and-Miss transform** searches for an exact match of the structuring element. It provides a simple form of template matching. We can also add don't care terms in the structuring element. It is given by:

$$H = I \otimes S$$

Thinning and *Thickening* is defined using *hit-and-miss transform*:

1. **Thinning**:
$$I \oslash S = I \backslash (I \otimes S)$$

2. **Thickening**
$$I \odot S = I \cup (I \otimes S)$$

Thinning and *Thickening* are dual operations:

$$(I \odot S)^C = I^C \oslash S$$

These operations are often performed in sequence (**Sequential Thickening/Thinning**) with different structuring elements:

$$I \oslash \{S_i : i = 1, \ldots, n\} = (((I \oslash S_1) \oslash S_2) \ldots S_n)$$

In sequential thinning sometimes the same strel is used in different just rotations. This is called the **Golay element/alphabet**:

L_1
0	0	0
*	1	*
1	1	1

L_2
*	0	0
1	1	0
*	1	*

L_3
1	*	0
1	1	0
1	*	0

L_4
*	1	*
1	1	0
*	0	0

$\ldots$ L_8
0	0	*
0	1	1
*	1	*

✠ The **Medial Axis Transform (MAT)** is also a skeletonization technique. It can be basically looked as starting a grass-fire at the boundaries, and the skeleton will be lines where two fire fronts meet. It can also be looked as the union of centers of maximal discs which touch atleast two boundaries simultaneously. Using the radius of the discs we can also order the skeleton pixels.

✠ Skeletonization can also be done through morphology using the structuring element:

$B =$
0	1	0
1	1	1
0	1	0

The n^{th} skeleton subset is:

$$S_n(X) = (X \ominus_n B) \, [(X \ominus_n B) \circ B]$$

The skeleton is the union of all skeleton subsets:

$$S(X) = \bigcup_{n=1}^{\infty} S_n(X)$$

Remember we can reconstruct X:

$$X = \bigcup_{n=1}^{\infty} S_n(X) \oplus_n B$$

We can reconstruct X from the MAT but not directly from $S(X)$.

✠ **Gray-level Morphology** (the image and the strel are grey level arrays) can be used for removing speckle noise, smoothing, etc. We define an *Umbra* $U(f)$ as filling all the pixels below the top surface, and the *top surface* $T(R)$ as the inverse of it. Now *erosion* is defined as:

$$f \ominus k = T(U(f) \ominus U(k))$$

and *dilation* as:

$$f \oplus k = T(U(f) \oplus U(k))$$

5 Image Filtering

✠ L is a **linear operation** if it preserves linear combinations:

$$L(\alpha I_1 + \beta I_2) = \alpha L(I_1) + \beta L(I_2)$$

Examples of linear operators:

1. **Correlation**:
$$I'(x, y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} K(i, j) I(x + i, y + j)$$
This is just like centering the kernel over the position (x, y)Correlation of kernel K with a unit impulse reveals a 180° rotated copy of the kernel.

2. **Convolution**:
$$\begin{aligned} I'(x, y) &= \sum_{i=-a}^{a} \sum_{j=-b}^{b} K(i, j) I(x - i, y - j) \\ &= \sum_{i=-a}^{a} \sum_{j=-b}^{b} K(-i, -j) I(x + i, y + j) \\ I' &= K * I \end{aligned}$$
So if $K(i, j) = K(-i, -j)$ then *correlation* will equal *convolution*.

✠ Types of filters (note that smoothing kernels should sum to 1 to avoid inducing any brightness change in the image; and sharpening kernels should sum to 0):

1. *Blurring* with **box filter / mean filter** (low-pass filter):

$$K = \frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

   If not using a box, and giving more weightage closer to the center will give a weighted smoothing filter.

2. *Sharpening* with **Laplacian filter**:

$$K = \begin{bmatrix}0 & 0 & 0\\ 0 & 2 & 0\\ 0 & 0 & 0\end{bmatrix} - \frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

3. **Gaussian Filter** - smoothing:

$$\begin{aligned} G_\sigma(x,y) &= \frac{1}{2\pi\sigma^2}e^{-(x^2+y^2)/2\sigma^2} \\ &= \left(\frac{1}{2\pi\sigma^2}e^{-x^2/2\sigma^2}\right)\left(\frac{1}{2\pi\sigma^2}e^{-y^2/2\sigma^2}\right) && \text{the mask is separable} \\ &= g(x)g(y) && \text{allows } I' = g(x) * (g(y) * I) \end{aligned}$$

   Its useful to set the gaussian filter's half-width greater than 3σ. A gaussian convolves by itself is a gaussian with std. dev of $\sigma\sqrt{2}$. Such repeated gaussian convolution gives us the *scale space* of the image. Important facts about the gaussian kernel: (1) single lobe (also in frequency domain); (2) neighbor influence decreases monotnoically.

Need to check what happens when filter falls off the edge of the image. Possible solutions: (1) clip filter, (2) wrap around, (3) copy edge, (4) reflect across edge, (5) vary filter at edge.

Smoothing in filtering is almost equivalent to integration; and sharpening is like differentiation.

Sharpening can also be looked at as the 1^{st} and 2^{nd} derivatives:

1. 1^{st} derivative:

$$\begin{aligned} 1D \quad &\Rightarrow \quad \frac{\partial f}{\partial x} = f(x+1) - f(x) \\ 2D \quad &\Rightarrow \quad \nabla f = \begin{bmatrix}\partial f/\partial x\\ \partial f/\partial y\end{bmatrix} = \begin{bmatrix}g_x\\ g_y\end{bmatrix} \\ &\qquad\quad |\nabla f| = \sqrt{g_x^2 + g_y^2}, \qquad \tan\alpha = g_y/g_x \end{aligned}$$

   The differential g_x or g_y can be computed by any differential operator †.

2. 2^{nd} derivative:

$$1D \quad \Rightarrow \quad \frac{\partial^2 f}{\partial x^2} = f(x+1) + f(x-1) - 2f(x)$$

$$2D \quad \Rightarrow \quad \nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \rightarrow \begin{cases} \frac{\partial^2 f}{\partial x^2} = f(x+1,y) + f(x-1,y) - 2f(x,y) \\ \frac{\partial^2 f}{\partial y^2} = f(x,y+1) + f(x,y-1) - 2f(x,y) \end{cases}$$

Here we can derive the **laplacian** kernel as: $K = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$

† We can approximate **differentation** by *convolution* with the kernel $K = \begin{bmatrix} -1 & 1 \end{bmatrix}$. The following are some other differential operators:

$$\text{Prewitt } K = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \qquad \text{Sobel } K = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

✠ *Sharpening* over an image can be done to increase high frequency components to enhance edges. It is usually done as: $I' = I + \alpha(K * I),\ \alpha \in [0,1]$. Hence *sharpening* can be done as $I' = I + \alpha\nabla^2 I$ which ∇^2 is the *Laplacian*. Similarly we can perform *unsharpening*: $I' = I + \alpha(I - K * I)$

✠ The **Fourier Domain** represents the image as the sum of weighted frequency components (mathematically represented as a sum of sines and cosines):

1. The continuous Fourier transform in 1D:

$$\mathcal{F}\{f(t)\} = F(\mu) = \int_{-\infty}^{\infty} f(t) e^{-j2\pi\mu t} dt$$

$$\mathcal{F}^{-1}\{F(\mu)\} = f(t) = \int_{-\infty}^{\infty} F(\mu) e^{j2\pi\mu t} d\mu$$

The inverse transform can be thought of as summing the contribution of signals with different frequencies (and amplitudes) to reconstruct a spatially and temporally periodic signal.

2. The **Discrete Fourier transform (DFT)** in 1D:

$$F(u) = \sum_{x=0}^{M-1} f(x)e^{-j2\pi ux/M}, \quad u = 0, 1, 2, \ldots, M-1$$

$$F^{-1} = f(x) = \frac{1}{M}\sum_{u=0}^{M-1} F(u)e^{j2\pi ux/M}, \quad x = 0, 1, 2, \ldots, M-1$$

Where M could be the length of a row of an image indexed by x. The function returned by DFT is indexed by u which represents the frequency of the sinusoidal component (remember euler's formula: $e^{j\theta} = \cos\theta + j\sin\theta$). DFT can be thought of a change of basis: we move from temporal space where signal is represented as coefficients of delta functions to frequency where signal is represented as coefficients of sinusoidal basis functions. The DFT in 2D is given by:

$$F(u, v) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x, y)e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}, \quad u = 0, 1, 2, \ldots, M-1, \quad v = 0, 1, 2, \ldots, N-1$$

The F returned as an $M \times N$ array of complex numbers. We usually plot $|F|$ (the magnitude / amplitude spectrum) and $\arg F$ (is the phase spectrum) as images. F is periodic: $F(u, v) = F(u + M, v) = F(u, v + N) = F(u + M, v + N)$; and has conjugate symmetry: $|F(u, v)| = |F(-u, -v)|$. The phase spectrum holds most of the information. The magnitude image usually shows that the lower frequencies (near the center pixel giving the DC component) contains more image information than the higher ones. The image would also show the dominating directions in the original image.

Convolution, now, can be done in Fourier domain by $\mathcal{F}[f * g] = \mathcal{F}[f]\mathcal{F}[g]$: (1) Compute DFT of f and g, (2) Multiply F and G (centering the smaller image on the bigger one), (3) compute inverse of FG. Filters can also be expressed in Fourier domain:

1. Ideal low pass filter:

$$K[u.v] = \begin{cases} 0 & u^2 + v^2 > r^2 \\ 1 & \text{otherwise} \end{cases}$$

The inverse of this would be ideal high pass filter.

2. Gaussian low pass filter:

$$K[u.v] = \exp\left[-\frac{u^2 + v^2}{2\sigma^2}\right]$$

$1 - K(u, v)$ would be Gaussian high pass filter.

✠ Non-linear filters include **Rank filters** which are based on picking values from the neighborhood according to grel levels:

1. **Max-filter**:
$$I'(m,n) = \max\{I(i,j) \,:\, (i,j) \in N(m,n)\}$$
2. **Min-filter**:
$$I'(m,n) = \min\{I(i,j) \,:\, (i,j) \in N(m,n)\}$$
3. **Median-filter**:
$$I'(m,n) = \text{median}\{I(i,j) \,:\, (i,j) \in N(m,n)\}$$
4. **Range-filter**:
$$I'(m,n) = \max\{I(i,j) \,:\, (i,j) \in N(m,n)\} - \min\{I(i,j) \,:\, (i,j) \in N(m,n)\}$$

6 Edge Detection

✠ **Undersampling** can have the effect of missing out on information. This can have adverse effects while trying to reconstruct the continuous signal back from the sampling. This results in **aliasing** - signals "traveling in disguise" as other frequencies. We can prevent *aliasing* by introducing low pass filter before sampling the signal e.g. a weighted moving average filter. We can, of course, increase the sampling rate making it above the *Nyquist Frequency*.

✠ **Image sub-sampling** is about throwing every other row and column to create 1/2 sized image. As said before, to achieve better results, do low-pass Gaussian filtering before sub-sampling. The filter size should double with each 1/2 size reduction.

✠ One way to **blend** two images is to simply weight them accordingly across the boundary.

✠ As discussed before we cab build a sharpening (edge detection) filter like a Laplacian. Because this kernel is essentially an approximation of the second derivative measurement on the image they are very sensitive to noise. To counter this, the image is often Gaussian smoothed before applying the Laplacian filter. This pre-processing step reduces the high frequency noise components prior to the differentiation step. We can combine the Gaussian filter with the Laplacian filter to give the **Laplacian of a Gaussian (LoG)** filter.

✠ A summary of the steps in **Laplacian pyramid coding and decoding**. First the original image g_0 is used to create images at different Gaussian pyramid levels $g_1, g_2, \ldots$. Levels of the Laplacian pyramid $L_0, L_1, \ldots$ are then computed as the differences between adjacent Gaussian levels. Laplacian pyramid elements are quantized to yield the Laplacian pyramid code $C_0, C_1, \ldots$. To reconstruct the image, we only need these codes - which is generated by summing levels of the code pyramid.

✠ The basic steps behind good edge detection techniques is: (1) noise reduction; (2) edge enhancement; (3) edge localization. One way to do this would be to simply compute the image gradients:

$$g_x(x, y) = f(x + 1, y) - f(x - 1, y)$$
$$g_y(x, y) = f(x, y + 1) - f(x, y - 1)$$

These can be represented by the *Prewitt* or *Sobel* kernels. The **Edge map** is a way to get the location of the edges, and it is created by thresholding the gradient magnitude: $|g| = \sqrt{g_x^2 + g_y^2} > \tau$

✠ **Canny Edge Detector** combines noise reduction and edge enhancement in the following steps:

1. Either smooth with Gaussian and differentiate OR convolve with the derivative of a Gaussian.

2. Quantizing the gradient direction, we keep only the gradient magnitudes which are local maximum in their gradient direction. We set all other gradient magnitudes to 0. This is called **Non-maximal suppression**.

3. Now we can simply threshold the *edge map*. Since it is nearly impossible to specify a single threshold dividing edges from non-edges, we use a low and a high threshold. Since we will certainly be sure about high thresholded pixels being edges, we will add 8-connected neighbors from the low thresholded edge map to this pixel. Iteratively doing this will build the complete edge-map. This method is called **Hysteresis thresholding**.

✠ **Marr-Hilderth Edge Detector**:

1. Uses second derivative of a Gaussian (called the *Laplacian of a Gaussian*):

$$LOG = \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) G(x, y)$$

2. Find zero crossings after convolving with LOG.

This seems to be quite sensitive to noise. Remember that the LOG can be approximated with the difference of two Gaussians, as illustrated in the scale space method:

$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k-1)\sigma^2\nabla^2 G$$

✠ **Model Fitting** is a method of finding edges by fitting a surface with an edge. The parameters for this model edge are usually (1) orientation, (2) position, (3) intensities either side of the edge. It finds least square fits in each small window. **Hough Transform** is one technique for model fitting:

1. At every edge pixel the equation of the local edge line is stored:

$$x\cos\theta + y\sin\theta = \rho$$

   only θ and ρ are stored.

2. By quantizing θ a histogram of lines is stored. Alternatively we can simply take the *radon transform* of an image which is a graph of ρ against θ (in this case the origin is the center pixel - while in *Generalized Hough transform* which is used for object detection the center pixel is taken as the center of the object).

3. Thresholding the graph gives us dominant lines in the image.

Unlike the first (*canny*) and second derivative (*marr-hilderth*) approach to edge detection, model fitting is not that sensitive to noise. Although it is slow.

✠ Edge-detector performance can be gauged by:

1. Probability of false edges
2. Probability of missing edges
3. Error in edge angle
4. Mean squared distance from true edge

7 Corner / Feature Detection

✠ Feature detection is important in a wide variety of applications including (1) stereo matching and 3D reconstruction, (2) tracking (find same features in next frame), (3) localization, (4) recognition. Corner/Feature/Interest points usually need to have the following characteristics:

1. They are distinctive
2. Stable from image to image
3. They are invariant to view point
4. Invariant to lighting conditions
5. Invariant to object deformations and partial occlusions
6. ... if possible, geometrically meaningful

✠ **Moravec Interest Operator** is quite simple:

1. Uses a window surrounding each pixel as a matching template
2. Uses *autocorrelation* (SSD) in the surrounding area of the pixel with the matching template:

$$\epsilon(\Delta x, \Delta y) = \sum_{(x,y)\in N(3,3)} w(x,y)(\underline{I(x+\Delta x, y+\Delta y)} - I(x,y))^2 \tag{1}$$

where w is the weight function. Hence good matches in any direction means that the region is flat. Good matches in one direction indicates an edge. Good matches in no direction marks a distinctive point feature. Hence **cornerness** measure could be the minimum ϵ in the neighborhood.

3. We can do *non-maximal suppression* i.e. set pixels which have an 8-neighbor with a higher *cornerness* to zero. This can be done efficiently by dilating the *cornerness* image.

4. In the end we threshold the *cornerness* image.

✠ To construct the **image structure tensor** we will simply generalize the underlined part in Equation 1 using Taylor expansion:

$$I(x+\Delta x, y+\Delta y) \approx I(x,y) + \begin{bmatrix} \frac{\partial I(x,y)}{\partial x} & \frac{\partial I(x,y)}{\partial y} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$$

Hence we can derive the *cornerness* value to be the following:

$$\begin{aligned} \epsilon(\Delta x, \Delta y) &= \sum_{x\in N}\sum_{y\in N} \left[\begin{bmatrix} \frac{\partial I(x,y)}{\partial x} & \frac{\partial I(x,y)}{\partial y} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \right]^2 \\ &= \sum_{x\in N}\sum_{y\in N} \begin{bmatrix} \Delta x & \Delta y \end{bmatrix} \left[\begin{bmatrix} \frac{\partial I(x,y)}{\partial x} \\ \frac{\partial I(x,y)}{\partial y} \end{bmatrix} \begin{bmatrix} \frac{\partial I(x,y)}{\partial x} & \frac{\partial I(x,y)}{\partial y} \end{bmatrix} \right] \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \\ &= \sum_{x\in N}\sum_{y\in N} \begin{bmatrix} \Delta x & \Delta y \end{bmatrix} \begin{bmatrix} \langle I_x^2 \rangle & \langle I_x I_y \rangle \\ \langle I_x I_y \rangle & \langle I_y^2 \rangle \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \mathbf{x}^{\mathrm{T}} \mathbf{S} \mathbf{x} \end{aligned}$$

The *image structure tensor* $\mathbf{S}$ is the **Harris matrix** where the angle brackets denote summation over the neighborhood. The **Harris Corner Detector** uses $\mathbf{S}$ since it captures the curvature of the local autocorrelation surface. we can reject unstable image structures (like edges instead of corners) by singular values of image structure tensor $\mathbf{S}$ (corner: both high eigenvalues; edge: one large one small; homogeneous: both small). The characteristic polynomial is given by:

$$\lambda^2 - \lambda\left(\langle I_x^2 \rangle + \langle I_y^2 \rangle\right) + \langle I_x^2 \rangle \langle I_x^2 \rangle - \langle I_x I_y \rangle^2 = 0$$

Since eigen-decomposition can be expensive, we simply compute:

$$C = |\mathbf{S}| / \mathrm{Trace}(\mathbf{S}) = \lambda_1 \lambda_2 / (\lambda_1 + \lambda_2)$$

Harris Corner Detector uses the C image to get the corner/feature points. After performing *non-maximal suppression* it thresholds the image. We can do Gaussian smoothing when computing I_x and I_y to eliminate multiple responses to the same corner.

Harris and Stephens combines edge and corner detection by using $R = |\mathbf{S}| - k\mathrm{Trace}^2(\mathbf{S})$. Using $0 < k < 0.25$ we get an R where edges as negative and corners as positive.

✠ **Univalue Segment Assimilating Nucleus (USAN)** uses a circular mask (whose center is the *nucleus*) to develop a template which lies within its circular boundary. The *USAN* is simply the ratio of area of the template which has the same intensity within a threshold as the *nucleus*. Note that at corners, *USAN* should be less than half. Hence we can describe a **Smallest USAN (SUSAN** for feature points.

✠ In **FAST (Feature from Accelerated Segmented Test)**, a feature is also found at each pixel by checking pixels that are at a given radius. Around each pixel p, if n contiguous pixels are all brighter than the nucleus by at least t or all darker than the nucleus by t, then the pixel under the nucleus is considered to be a feature. The radius is set to 3 (corresponding to a circle of 16 pixels circumference), and n is 9. The values associated with darker, similar and brighter are d, s, b. Now we train a decision tree to maximize information gain:

$$H(P) - H(P_d) - H(P_s) - H(P_d)$$

✠ **SIFT - scale invariant feature transform** is a method to find robust image keypoints. This is done by encoding the properties of the neighborhood in a descriptor vector. We can then match points defining an appropriate distance metric.

It leverages the idea that images have different structures at various different scales:

1. Make a **scale space** which is simply a stack of images (the original one at the bottom) produced by convolving the top-most image in the stack by a fixed Gaussian (blurring) - these represent structures at increasingly large scales.

2. Since image gradients are more stable features than raw luminance, we take the difference of consecutive images in the scale space (**DOG - difference of Gaussians**):

$$\begin{aligned} & & G(x,y,\sigma) &= \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \\ \text{DOG} \quad & \rightarrow \quad & D(x,y,\sigma) &= G(x,y,\sigma) * G(x,y,\sigma) * I(x,y) - G(x,y,\sigma) * I(x,y) \\ & & &= (G(x,y,k\sigma) - G(x,y,\sigma)) * I(x,y) \end{aligned} \tag{2}$$

   We can do step 2 because both convolution and difference are linear operators.

3. Now find the extrema in DOG space i.e. points larger (or smaller) than 26 neighbors in the DOG stack.

4. Now we need localize keypoints to subpixel accuracy, which is done by Taylor expansion up to quadratic terms of the scale space function $D(x, y, \sigma)$ around the potential keypoint (taking $\mathbf{x} = (x, y, \sigma)$):

$$D(\mathbf{x}) = D + \frac{\partial D^{\mathrm{T}}}{\partial \mathbf{x}}\mathbf{x} + \frac{1}{2}\mathbf{x}^{\mathrm{T}}\frac{\partial^2 D}{\partial \mathbf{x}^2}\mathbf{x}$$

where D and its derivatives are evaluated at the keypoint and $\mathbf{x}$ is the *offset* from this keypoint. Taking the derivative of this function with respect to this offset and setting the resulting expression to zero allows us to estimate the position of the keypoint up to sub-pixel accuracy:

$$0 = \frac{\partial D^{\mathrm{T}}}{\partial \mathbf{x}} + \frac{\partial^2 D}{\partial \mathbf{x}^2}\hat{\mathbf{x}}$$

$$\begin{bmatrix} \frac{\partial^2 D}{\partial x^2} & \frac{\partial^2 D}{\partial yx} & \frac{\partial^2 D}{\partial \sigma x} \\ \frac{\partial^2 D}{\partial yx} & \frac{\partial^2 D}{\partial y^2} & \frac{\partial^2 D}{\partial \sigma y} \\ \frac{\partial^2 D}{\partial \sigma x} & \frac{\partial^2 D}{\partial \sigma y} & \frac{\partial^2 D}{\partial \sigma^2} \end{bmatrix} \begin{bmatrix} x \\ y \\ \sigma \end{bmatrix} = - \begin{bmatrix} \frac{\partial D}{\partial x} \\ \frac{\partial D}{\partial y} \\ \frac{\partial D}{\partial \sigma} \end{bmatrix}$$

$$\hat{\mathbf{x}} = -\frac{\partial^2 D^{-1}}{\partial \mathbf{x}^2}\frac{\partial D}{\partial \mathbf{x}}$$

5. Use of *image structure tensor* (see above) to reject unstable image structures:

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

6. Gradients binned by orientation to form a histogram, and orientation of feature is taken as the peak of the histogram (if more than one peak exists, split into two keypoints with different orientations). Since the descriptor is computed relative to this orientation, it achieves rotational invariance.

✠ Now the task comes down to finding the keypoint *correspondence*. Once *SIFT* gives us stable feature points, they need to be characterized by their appearance. This is done by a histogram of image gradients in a region around a keypoint:

1. Calculate the image gradients around a keypoint, weighted by a gaussian.
2. Pool over a space to form 8 bin orientation histograms in a 4×4 grid.
3. The values of histograms are concatenated to form a 128 elements feature vector f $(8 \times 4 \times 4)$. Normalize to unit length to make them invariant to changes in contrast.

The feature vector f is matched to feature vectors $\mathbf{F}'$ in the other image around a certain region. The similarity measure is the Euclidean distance between the SIFT descriptors.

8 Feature Characterization

✠ **Region/Spatial moments** for a region R in the image I:

$$m_{pq} = \sum_{(x,y)\in R} x^p y^q I(x,y)$$

The order of the moment is $p+q$. So m_{00} is simply the area. The normalized 1^{st} **order moment** of a variable ($pq = 10$ or 01) is the expectation:

$$\bar{x} = \frac{m_{10}}{m_{00}}, \qquad \bar{y} = \frac{m_{01}}{m_{00}}$$

The **central moment** is given by:

$$\mu_{pq} = \sum_{(x,y)\in R} (x-\bar{x})^p (y-\bar{y})^q I(x,y)$$

1. 0^{th} order *central moment* is the Area: $\mu_{00} = m_{00}$
2. 1^{st} order *central moment* which is equal to 0!
3. 2^{nd} order *central moment* is called the variance (remember $\sigma^2 = var$). Also μ_{11} gives covariance.
4. 3^{rd} order *central moment* is **skewness** - tells how lopsided is the region.
5. 4^{th} order *central moment* is **kurtosis** - tells if the region is skinny/tall or short/wide.

To get scale-invariance, we can use **normalized central moments**:

$$\vartheta_{pq} = \frac{\mu_{pq}}{(\mu_{00})^\alpha}, \qquad \alpha = \frac{p+q}{2} + 1$$

This is leads us to **Hu's moment features**, denoted by φ, which are invariant to translation, scale and rotation:

$$\begin{aligned}
\varphi_1 &= \vartheta_{20} + \vartheta_{02} \\
\varphi_2 &= (\vartheta_{20} - \vartheta_{02})^2 + 4\vartheta_{11}^2 \\
\varphi_3 &= (\vartheta_{30} - 3\vartheta_{12})^2 + (3\vartheta_{21} - \vartheta_{03})^2 \\
\varphi_4 &= (\vartheta_{30} + \vartheta_{12})^2 + (\vartheta_{21} + \vartheta_{03})^2
\end{aligned}$$

✠ The are also a list of **Affine invariants**, where the first one is:

$$A_1 = [\mu_{20}\mu_{02} - \mu_{11}^2]/\mu_{00}^4$$

Other region descriptors include:

1. Direction: $\theta = \frac{1}{2}\tan^{-1}\left(\frac{2\mu_{11}}{\mu_{20}-\mu_{02}}\right)$
2. Rectangularity: max ratio of R's area to its smallest bounding rectangle.
3. **Euler number**: #Connected Components − #Holes

✠ Boundary features include:

1. **Perimeter** $P(R)$ i.e. the length of the boundary of region R.
2. **Compactness** $C(R) = m_{00}/P(R)^2$
3. **Boundary coding** schemes:
   (a) List of pixels
   (b) **Chain codes**: Rather than encoding each pixel value, we can store just the origin and the path given by the direction from the previous point. The **Freeman's code** gives the direction encoding scheme:

	1	
2		0
	3	

3	2	1
4		0
5	6	7

   Steps for encoding boundary:
   (1) Start with leftmost top pixel
   (2) Follow the boundary anti-clockwise
   (3) Stop when we start repeating the path
   (c) **Fourier Descriptors**: a boundary can be represented by $U = \{(x_n, y_n) : 1 \leq n \leq N\}$. Now the coordinates x and y are periodic functions of n : $u(n) = x(n) + jy(n)$, hence we can represent them by the frequency components of x and y. So using DFT we generate the fourier descriptors a_n and b_n, $n = 0, \ldots, N-1$:

$$a_n = \frac{1}{N}\sum_{k=0}^{N-1} x_k e^{-j2\pi nk/N}$$

$$b_n = \frac{1}{N}\sum_{k=0}^{N-1} y_k e^{-j2\pi nk/N}$$

To get rotation invariance we can use $r_n = \sqrt{|a_n|^2 + |b_n|^2}$; and scale invariance by $w_n = r_n/r_1$. It is worth noting that the lower n components encode broad shape characteristics while the higher components encode detail. Scaling just requires multiplying coefficients by α and rotation requires multiplying coefficients by $\exp(j\theta)$.

✠ Traditional vectorization involves fitting curves to binary raster line images. **Object-based vectorization** creates and renders meshes extracted from individual objects contained in grayscale or color raster images.

✠ **GLOH (Gradient Location and Orientation Histogram)** is a robust image *SIFT*-like descriptor that considers more spatial regions for the histograms. Unlike *SIFT* it uses a polar location grid for developing a histogram of quantized orientations. The higher dimensionality of the descriptor is reduced to 64 through principal components analysis (PCA).

✠ **Shape Context** also uses an orientation histogram: (1) compute the perimeter points of the shape (subsample?), (2) compute log-polar graph for each point ($\log r$ against θ), (3) normalize for average scale. Matching can be done by *shape context* by establishing correspondences between the points for two shapes using the *Hungarian method* - with weights given by the sum of histogram dissimilarity and tangent angle dissimilarity.

✠ **HOGs (Histogram of Oriented Gradients)** can be used for shape matching. (1) Compute the average gradient image from the training images, (2) Divide image into blocks and compute maximum positive SVM and negative SVM weights, (3) count occurrences of gradient orientation in localized portions, (4) weigh them by the SVM weights. Found that fine-scale gradients, fine orientation binning, and relatively coarse spatial binning works best for feature localization.

✠ **Textons** are fundamentally micro-structures in natural images. We can learn these *textons* from texture images. An image is a superposition of image bases selected from an over-complete dictionary of Gabor (and maybe even Laplacian of Gaussian) functions at various locations, scales, and orientations. This is called the **Gabor-filter bank**. In short we can process images for features based on *textons* - where *textons* are filter responses of the *Gabor-filter bank*.

9 Color

✠ Humans photoreceptors are neurons that absorb photons and signal the brain - come in two flavors - 120 million **rods** (best for motion perception) and 6-7 million 3 types of **cones** for color perception. These photoreceptors send input to photoreceptive **ganglion cells**.

✠ **Metamers** are light spectra with the same tristimulus response i.e. they are mixture of wave-

lengths which induce the same response in the human eye.

✠ Color cameras:

1. **Prism color camera** separates light into 3 beams using dichroic prism. It requires 3 sensors for capturing the separated light.
2. **Filter mosaic** is to coat filter directly onto the sensor (like a bayer pattern) - then demosaic to obtain the full color image in full resolution. This gives us our usual CCD cameras.
3. **Filter wheel** works by rotating different filters infront of the lens - but only suitable for static scenes.
4. **CMOS Foveon's X3** works by sampling all three colors at the same pixel.

✠ **White balance** is the process of removing unrealistic color casts, so that objects which appear white in person are rendered white in the image. This variation can happen under different lighting conditions (remember the color signal is a product of the illumination and reflectance).

✠ For color imaging, measurement from the $k-th$ sensor at pixel (x, y) is:

$$I_k(x, y) = \int_0^\infty R_k(\lambda) L(x, y, \lambda) d\Lambda \tag{3}$$

where L is the scene radiance/image irradiance, and R_k is the response function of the k^{th} sensor. The **integrated white condition** is satisfied if $\int_0^\infty R_k(\lambda) d\lambda$ is equal for each k, in which case each I_k is equal for white light.

✠ There are two primary kinds of color models:

1. **Additive model**: adds light and has RGB as their primary colors. (remember $R + B =$ magenta; $B + G =$ cyan; and $G + R =$ yellow
2. **Subtractive model**: adds pigments (*subtracts* light) and has Cyan (C), Magenta (M) and Yellow (Y) as their primary colors:

$$\begin{aligned} C &= B + G = 1 - R \\ M &= R + B = 1 - G \\ Y &= G + R = 1 - B \end{aligned}$$

We can also describe colors perceptually:

1. **Brightness / Lightness / intensity**: is the chromatic equivalent of overall intensity:

$$I = R + G + B$$

2. **Hue**: describes the dominant wavelength.

$$H = \tan^{-1}\left(\frac{\sqrt{3}(G-B)}{(R-G)+(R-B)}\right)$$

3. **Saturation**: refers to purity of the dominant wavelength:

$$S = 1 - \left(\frac{3\min\{R,G,B\}}{R+G+B}\right)$$

4. **Chromaticity**: combines *hue* and *saturation*

Remember RYGCBM hexagon. **Normalized color**: $(r, g, b) = (R/I,\ G/I,\ B/I)$, removes dependence on geometry.

HSI(L) can be described by the cylinder where the angle is given by the *hue*, the distance from the center is given by *saturation* and going up the cylinder increases the *lightness/intensity* (top of the cylinder is white, bottom is black).

HSV can be described by the cylinder where the angle and distance is similarly defined, and going up the cylinder increases the *value*.

✠ **CIE** primaries define primary colors as XYZ, by which we can produce any color as a linear combination (positive coefficients) of XYZ pure colors. *CIE* diagrams represents all chromaticities visible to the eye. These are shown in color and this region is called the **gamut** of the human eye (is tongue-shaped). The curved edge of the *gamut* is called the spectral locus and corresponds to monochromatic light (fully saturated colors), with wavelengths listed in nanometers. The straight edge on the lower part of the gamut is called the line of purples. These colors, although they are on the border of the gamut, have no counterpart in monochromatic light. Less saturated colors appear in the interior of the figure with white at the center. It has the following interesting properties: (1) a line joining any two points, then any color along the line can be given by the mixture of colors at the two points; (2) this idea can be extended to triangle (in fact to any convex shape); (3) Any line from the locus to the center is of constant hue; (4) a set of colors a device can disply is given by the triangle of its primary colors.

✠ In **Shafer's dichromatic reflectance model**, *scene radiance/image irradiance* has two components:

$$L(\lambda) = m_s c_s(\lambda) + m_b c_b(\lambda)$$

c_s is the surface reflected component, c_b is the body reflected component, and m_s and m_b are scale factors depending on illumination, view direction, and surface orientation:

$$c_s(\lambda) = e(\lambda)$$
$$c_b(\lambda) = \sigma(\lambda)e(\lambda)$$

where σ is the surface reflectance, and e is the *scene irradiance (illumination)*. Now Equation 3 can be replaced by:

$$\mathbf{C} = m_s\mathbf{C}_s + m_b\mathbf{C}_b$$
$$\text{where} \quad \mathbf{C}_s = \int \mathbf{R}(\lambda)e(\lambda)d\lambda$$
$$\mathbf{C}_b = \int \mathbf{R}(\lambda)\sigma(\lambda)e(\lambda)d\lambda$$

Planar cluster: m_b and m_s vary over a surface with fixed σ. For *Lambertian* surfaces $m_b = \cos\theta$ where θ is the angle of incidence. Often m_s is non-zero for only a few orientations of the surface. For *matte* materials $m_s = 0$, and $\mathbf{C}$ lies on the diffuse reflectance line. For *shiny* materials $\mathbf{C}$ lies on the specular line, since $m_b = 0$

✠ Humar color perception is independent of illumination color (i.e. we see surface reflectance rather than scene radiance). Correcting colors for illumination is called **color constancy**. If we suppose $R_k(\lambda) = \delta(\lambda_k)$ and matte reflection ($m_s = 0$), then $(R, G, B) \to (\alpha R, \beta G, \gamma B)$ under change in illumination (where (α, β, γ) will be the illumination color). To correct this we do **illumination correction**:

$$R' = \frac{NR}{\sum R}, \qquad G' = \frac{NG}{\sum G}, \qquad B' = \frac{NB}{\sum B}$$

To bring invariance to illumination color and intensity:

$$R' = \frac{R_1G_2}{R_2G_1}, \qquad G' = \frac{G_1B_2}{G_2B_1}, \qquad B' = \frac{B_1R_2}{B_2R_1}$$

L*,a*,b* is derived from *CIE*, which ensures that any change in color value, brings an equivalent change according to visual importance (uncorrelated color space).

10 Matching and Flow

✠ Usually descriptors are evaluated based on a *recall* against *precision* graph:

$$\text{Recall} = \frac{TP}{TP + FN}$$
$$\text{Precision} = \frac{TP}{TP + FP}$$

✠ **Chamfer Matching** is an idea for model matching. Here we are given a model binary image and an input image. (1) we get the edges of the input image (after pruning week edges), (2) then we get the euclidean distance transform of the *edge map*, (3) the binary model is shifted over the distance transform image and at each shift position, the sum of distances at the model pixels is determined:

$$\sum_{i=1}^{n}\sum_{j=1}^{m} T(i,j)D(i+k,j+l)$$

and the shift position producing the smallest sum is chosen as the best-match position of the model in the image.

✠ **Hausdorff Distance** is a measure which ranks each point in A (the binary template) based on closeness to a point in B (the input binary image): and the measure is the most mis-matched point:

$$\begin{aligned} H(A,B) &= \max(h(A,B),\, h(B,A)) \\ \text{where} \quad h(A,B) &= \max_{a\in A}\min_{b\in B} \parallel a-b \parallel \end{aligned}$$

Another method for template matching is **Cross-correlation**:

$$S(x,y) = \sum_{(u,v)\in N_{m\times n}} I(x+u,\, y+v)T(u,\, v)$$

The problem with cross-correlation technique is that if I varies with position, bright patch will beat the right patch; also range of S depends on the size of the template. To correct for this we use **Normalized cross-correlation**:

$$C(x,y) = \frac{\sum_{(u,v)\in N_{m\times n}}[I(x+u,\, y+v)-\bar{I}][T(u,\, v)-\bar{T}]}{\sqrt{\sum_{(u,v)\in N_{m\times n}}[I(x+u,\, y+v)-\bar{I}]^2 \sum_{(u,v)\in N_{m\times n}}[T(u,\, v)-\bar{T}]^2}}$$

Other robust metrics include:

$$d_\alpha(x,y) = \left(\sum_{(u,v)\in N_{m\times n}} |I(x+u,\, y+v)-T(u,\, v)|^\alpha\right)^{1/\alpha}$$

$\alpha = 1$ is SAD, $\alpha = 2$ is SSD, larger α makes d more sensitive, and as $\alpha \to \infty$ this become more like maximum distance in N.

✠ **Optical Flow** is the pattern of apparent motion of objects, surfaces, and edges in a visual scene caused by the relative motion between an observer (an eye or a camera) and the scene itself. It is a dense form of tracking i.e. where each individual pixel went. It makes two simplifying assumptions:

1. The **Brightness constancy assumption (BCE)**, says that the brightness value doesn't change for each pixel:
$$I(x,\, y,\, t) = I(x + dx,\, y + dy, t + dt)$$

2. Nearby points move in a similar fashion.

Using the *BCE*, we can use the Taylor expansion to write:

$$I(x + dx,\, y + dy,\, t + dt) = I(x,\, y,\, t) \;+\; I_x dx + I_y dy + I_t dt + \underbrace{\text{H.O.T}}_{\text{terms we will ignore}}$$

Hence if dx and dy are small, the *brightness constancy assumption* will hold and we can simplify this to:

$$-I_t = I_x \frac{dx}{dt} \;+\; I_y \frac{dy}{dt} \tag{4}$$

Note that $\frac{dx}{dt}$ and $\frac{dy}{dt}$ are the speeds of the pixels. Note that there is only one equation to solve for these two unknowns. We can only get the perpendicular least squares component but we can't compute the parallel component. This can be related to the **Aperture problem** since we are only looking in a small neighborhood: the component of the motion field orthogonal to the spatial image gradient is not constrained by the BCE. The **Lucas-Kanade Method** approaches this problem (to solves the equation 4) by over constraining it; since we can suppose that the motion would be the same in a neighborhood we can write:

$$\begin{bmatrix} -I_{t1} \\ \vdots \\ -I_{t9} \end{bmatrix} = \begin{bmatrix} I_{x1} & I_{y1} \\ \vdots & \vdots \\ I_{x9} & I_{y9} \end{bmatrix} \begin{bmatrix} \frac{dx}{dt} \\ \frac{dy}{dt} \end{bmatrix}$$

Since the motion could be large, the second order terms could dominate. But we threw them out! Solution: do a *coarse-to-fine estimation* i.e. make an image pyramid of the two frames, and then solve the for $\frac{dx}{dt}$ and $\frac{dy}{dt}$ at each level and propogate the speeds as we go down the pyramid.