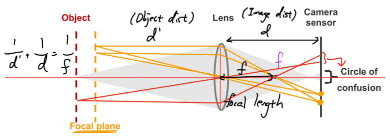
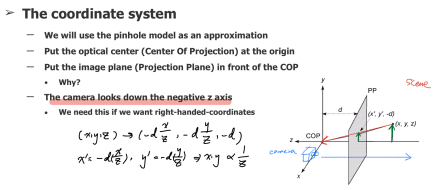
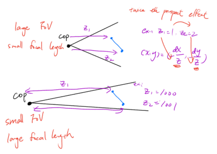
**Camera and Image Formation:** image distance(d): from pinhole(COP) to the sensor. If image distance is double => projected object height is doubled, but the amount of light is divided by 4. Pinhole camera image Formation will be upside-down. If the result is blurry => shrink the aperture (pinhole). However, shrinking aperture results in less light and also introduce diffraction (make it blur again). Better Approach=> We can make aperture larger and use lens to focus the light (Have a specific distance at which the objects are in focus, others projects to a circle of confusion). **Depth of field** => the area where the image can appear sharp. We can increase the Depth of field by reducing the size of the aperture. $D = \frac{f}{N}$, where $f$ is the focal length, $N$ is the f-number and $D$ is the aperture diameter. Area of the aperture $A = \pi(\frac{D}{2})^2 \propto \frac{1}{N^2}$. N usually = 2, 2.8, 4.0 …so that area = $\frac{A}{4}, \frac{A}{8}, \frac{A}{16} \ldots$.**Field of View:** The angle through which the camera can see the world. $f\uparrow, FoV\downarrow$. $FoV = 2\arctan(\frac{h}{2f})$ , where $h$ is the sensor size and $f$ is the focal length. **Effect if sensor size (EOS):** similar effect as cropping. $EOS\uparrow, Area\uparrow$. If focal length increases => subject remains the same size but background become closer, also have weaker perspective effect. Exposure: lens area x shutter time. These two have the same exposure (F5.6, 1/30 sec), (F11, 1/8 sec). Rolling shutter vs Global shutter => Rolling (cheaper, distorted on fast moving things), Global (snapshot of the scene at at single instant).




➢ The coordinate system
- We will use the pinhole model as an approximation
- Put the optical center (Center Of Projection) at the origin
- Put the image plane (Projection Plane) in front of the COP
  - Why?
- The camera looks down the negative z axis
  - We need this if we want right-handed-coordinates

$(x,y,z) \rightarrow (-d\frac{x}{z}, -d\frac{y}{z}, -d)$
$x' = -d(\frac{x}{z}),\ y' = -d(\frac{y}{z}) \Rightarrow x,y \propto \frac{1}{z}$

y
PP
Scene
d
(x', y', -d)
(x, y, z)
z
COP
camera
x

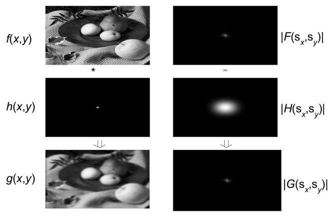


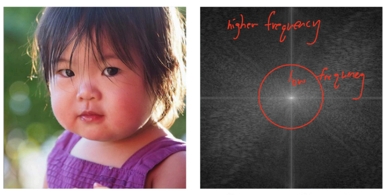
**Color: CIE RGB** Color Matching function => how much of each CIE RGB primary light must be combined to match a monochromatic light of wavelength given on x-axis (can be be negative, which means we need to add light to the test color's side). **CIE XYZ** => Instead of using primary lights RGB, we use imaginary primary lights (XYZ) to avoid having negative coefficient. **Chromaticity =>** normalization of XYZ color: x = X/(X+Y+Z)
**Automatic white balance:** 1. Make average color of scene to grey (Grey World Approach) 2. Make brightest object to white (White World Approach).

**Sampling, Frequency, and Filtering:**
Aliasing: in image (ex moire pattern, under-sample images), in video (wagon-wheel effect, when camera shutter speed to slow) => Fast-changing signals (high frequency) sampled too slowly. To solve this: we can perform pre-filtering. Why under sampling results in aliasing? Higher frequency components of the signal can "fold" back into the lower frequency range of the sampled signal, causing undistinguishable between low and high frequency. Why pre-filtering reduces aliasing? Pre-filter like low-pass filters remove high-frequency components from the signal. **Nyquist Theorem:** the sampling rate should be equal to or greater than twice the highest frequency in the signal. Aliasing occurs when a signal is sampled below its Nyquist sampling rate.
**Fourier Transform:** Represent a function as a weighted sum of sines and cosines. We can convert an image from spatial domain to frequency domain.
$f = A_0 f_0 + A_1 f_1 \ldots f_i = a\cos(2\pi\omega x)$ or $a\sin(2\pi\omega x)$. $\omega$ is the frequency.

**Filters:** 1.Weighted moving average (Box filter = $\frac{1}{9}\begin{bmatrix}1,1,1\\1,1,1\\1,1,1\end{bmatrix}$) 2.Gaussian Filter: $G_\sigma = \frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}} = \frac{1}{16}\begin{bmatrix}1,2,1\\2,4,2\\1,2,1\end{bmatrix}$, $\sigma$ increases => more blurry. Rule of thumb: set the filter half-width to about $3\sigma$.
**Convolution:** commutative $a * b = b * a$, associative $a*(b*c) = (a*b)*c$, distributive $a*(b+c) = a*b + a*c$ => Convolving twice with the Gaussian kernel with $\sigma$ is equal to convolving once with $\sqrt{2}\sigma$. Convolution Theorem: The Fourier transform of the convolution of two functions is the product of their Fourier transformation $F[g*h] = F[g]F[h]$ => Convolution in spatial domain equals to multiplication in frequency domain. The effect of a square filter is less natural compared to Gaussian blurring. In nature, light and shadows do not usually have hard boundaries but blend smoothly.
Fill factor: ratio of light sensitive area to total pixel area, lower may introduce aliasing.

**Edge detection:** $\nabla f = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}]$, gradient direction is given by $\theta = \tan^{-1}(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x})$. Edge strength is the magnitude $|\nabla f| = \sqrt{(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2}$. What if we have a noise signal => smooth first then find the edge ($\frac{d}{dx}(f*h) = (\frac{d}{dx}h)*f$, find the derivative of gaussian filter $h$.
Ex: Sobel Filter: vertical edge $\begin{bmatrix}-1,0,1\\-2,0,2\\-1,0,1\end{bmatrix}$. When on borders: convolve type: **full**: compute as long as at least one pixel overlaps. **Same**: compute conv when center pixel of the filter overlaps the image. **Valid**: all pixels must overlap with the filter
**Hybrid Images**: Low-Frequency image + High-Frequency image => larger scale: sees the high-frequency image, smaller scale: sees the low-frequency image





Spatial Domain　Frequency Domain

**Gaussian Pyramid:** For half-sizing images, if we directly perform subsampling, it will introduce aliasing (high frequency signal get sampled slowly) => Get rid of high frequencies (i.e. low-pass filter the image then subsample). The resize function in Python do this automatically (filtering + subsampling). The size of the Gaussian Pyramid is roughly $\frac{4}{3}$ of the original image ($A = S + \frac{S}{4} + \frac{S}{16} + \ldots = \sum_{i=0}^{n}\frac{S}{4^i}$, $\frac{A}{4} = \frac{S}{4} + \frac{S}{16} + \frac{S}{64}\ldots$ => $A \approx S + \frac{1}{4}A$ => $A \approx \frac{4}{3}S$). Good for search over translation (i.e. search in smaller size image then fine-tune at higher scales) and search over scales. $G = [G_1, G_2, G_3, \ldots, G_n]$ where $G_1$is the original image.
**Laplacian Pyramid:** High frequency part of the image. Two ways of calculating it: $L_k = G_k - f * G_k$ or $L_k = G_k - \text{expand}\, G_{k+1}$ . $L = [L_1, L_2, \ldots, L_n]$, where $L_n = G_n$ (the last layer in laplacian pyramid equals the last layer in gaussian pyramid to preserve rest informations). Collapsing the pyramid gets the original image. Expand operator: Create an image $G'_{k-1}$ => Fill $G'_{k-1}$ with a stride of 2 using pixels in $G_k$ => Filter $G'_{k-1}$ with Gaussian filter => scale the intensity by 4.
The Gaussian Pyramid and Laplacian Pyramid can be used for **image blending**: $I_{blend} = \alpha I_{left} + (1-\alpha)I_{right}$. Input: Image X, Y, and a mask A. Approach: 1. Build the Laplacian pyramid LX and LY from X and Y => 2. Build a Gaussian Pyramid GA from the mask A. => 3. Form a combination pyramid from LX and LY using the corresponding levels of GA as weights, i.e. $LBlend(i,j) = GA(i,j)*LX(i,j) + (1 - GA(i,j))*LY(i,j)$ => 4. Collapse LBlend to get the blended image. However this image blending methods may suffer from bad results if the two images have strong disparity in intensity (color mismatch between X and Y).

**Image Blending:** Different from Gaussian image blending, we want the content to blend in more seamlessly => Do editing in the gradient domain.
**Membrane interpolation:** $\min_f \iint_\Omega |\nabla f|^2$ with $f|_{\partial\Omega} = f^*|_{\partial\Omega}$. $\Omega$ is the region we want to interpolate, $\partial\Omega$ indicates the border of the in-fill region. $f$ and $f^*$ are the intensity of the in-fill region and original source image respectively. $f|_{\partial\Omega} = f^*|_{\partial\Omega}$ indicates that the intensity remains the same on the border of $\Omega$. In 1D example, the interpolation is basically linear interpolation between the boundary values. (Second derivative should be 0; filter=(-1,2,-1)). Solve for $Af = b \Rightarrow f = A^{-1}b$.

**Seamless Poisson Cloning:** Now, instead of minimizing the gradient to 0, we minimize to a gradient where $f$ is the composite image, $S$ is the source image and $T$ is the target image. $\min_f \iint_\Omega ||\nabla f(x,y) - \nabla S(x,y)||^2 \, dx\,dy$, subject to $f(x,y)|_{\partial\Omega} = T(x,y)|_{\partial\Omega}$. We can use Euler-Lagrange equation to optimize it, and the solution is $\begin{cases} \Delta f = \Delta S, & \text{if in } \Omega \\ f(x,y) = T(x,y), & \text{otherwise} \end{cases}$. $\Delta f(x,y) = f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4 \times f(x,y)$ and $\Delta S(x,y) = S(x+1,y) + S(x-1,y) + S(x,y+1) + S(x,y-1) - 4 \times S(x,y)$.

**Mixing Gradient:** $\nabla I(x,y) = \begin{cases} \nabla T(x,y)\text{ , if } ||\nabla T(x,y)|| > ||\nabla S(x,y)|| \\ \nabla S(x,y)\text{ , otherwise} \end{cases}$, if the gradient magnitude of the target image is greater than that of the source image, the target image's gradient is then employed. $4 \times f(x,y) - f(x+1,y) - f(x-1,y) - f(x,y+1) - f(x,y-1) =$

$(I(x,y) - I(x+1,y)) + (I(x,y) - I(x-1,y)) + (I(x,y) - I(x,y+1)) + (I(x,y) - I(x,y-1)$, where $I(x,y) - I(x+1,y) = \begin{cases} T(x,y) - T(x+1,y)\text{ , if } |T(x,y) - T(x+1,y)| > |S(x,y) - S(x+1,y)| \\ S(x,y) - S(x+1,y)\text{ , otherwise} \end{cases}$

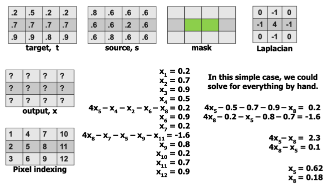


Image Processing=> two categories: **Point Processing** $g(x) = h(f(x))$ (change range of image) , and **Image Warping** $g(x) = f(h(x))$ (change domain of image).

**Point Processing examples:** 1. Gamma Correction: $s = r^\gamma$ (r, s are the intensity of grayscale images.2. Histogram Equalization: Probability Theory, 1D Transformation Formula: If $s = T(r)$, then $p_s(s) = p_r(r)|\frac{dr}{ds}|$, where $p_s(s)$ and $p_r(r)$ are the pdf of intensity s and r respectively. We can define T(r) as $s = T(r) = (L-1)\int_0^r p_r(w)d(w)$ => Since $\frac{ds}{dr} = (L-1)\frac{d}{dx}[\int_0^r p_r(w)dw] = (L-1)p_r(r)$, we have $p_s(s) = p_r(r)|\frac{1}{\frac{ds}{dr}}| = \frac{1}{L-1}$.

**Image Warping examples:** Linear Transformation (Scale, Rotation, Shear, Mirror…) => Properties: 1. Origin maps to origin 2. lines map to lines 3. parallel lines remain parallel 4. Satisfies $T(ax+by) = aT(x) + bT(y)$. 2D Transformations matrices: ex: Rotation(counter-clockwise) $R(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$

Affine Transformation $A = \begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix}$ => Properties: 1. Origin may not maps to origin 2. Lines map to lines 3. Parallel lines remain parallel. 4. Ratio of parallel line segments are preserved. Need 3 points correspondences to find (DOF = 6).

Projective (Perspective) Transformation $P = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{pmatrix}$ => Properties: 1. Origin may not maps to origin 2. Lines map to lines 3. Parallel lines may not remain parallel. 4. Ratio of parallel line segments are not preserved. Need 4 points correspondences to find (DOF = 8).

**Forward Warping:** Given transformation $T(x,y)$ and source image $f(x,y)$, send each pixel $f(x,y)$ to its corresponding position $(x',y') = T(x,y)$. If pixel lands between two pixels, do splatting (distribute color among neighboring pixels of $(x',y')$ **Inverse Warping:** Given transformation $T(x,y)$ and output image $g(x',y')$, send each pixel back $(x,y) = T^{-1}(x',y')$. If pixel lands between two pixels, do interpolation (nearest neighbor, bilinear, … ). Bilinear Interpolation: $f(x,y) = (1-a)(1-b)f[i,j] + a(1-b)f[i+1,j] + (ab)f[i+1,j+1] + ((1-a)b)f[i,j+1]$, where $a = x - i$ and $b = y - j$.

Mosaics: 1. collect correspondences (at least 3 for Affine, 4 for Perspective; however the more the better) => 2. Use Least squares to solve for homography matrix H $\begin{pmatrix} x' \\ y' \\ w' \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{pmatrix}\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$, $x' = \frac{ax+by+c}{gx+hy+1}$ => $ax + by + c - xx'g - yy'h = x'$, similar for y; therefore, each correspondences contributes to two equations. Solve for $\min|Ah - b|^2$, where $A \in R^{N\times 8}$, $h = [a,b,c,d,e,f,g,h]^T$ and $b = [x'_1, y'_1, \dots, x'_n, y'_n]^T$ => 3. Warp the content from one image to the other (from im1 into im2). First we check where will the corners in im1 maps to im2. For every coordinates within the warpped range of im1 on im2, we want to look up the colors from im1. This can be done by mapping the coordinate at (x',y') back to im1 with $H^{-1}$ and query the colors => 4. Overlay im2 content onto the wrapped im1 content.

Automatic Image Alignment: 1. Feature-based alignment (Find a few matching features and compute alignment) 2. Direct (pixel-based) alignment: search for alignment where more pixels agree.

**Feature-Based alignment:** Feature Detection (find important features, descriptors) => Feature Matching => Compute image transformation (RANSAC)
Descriptors: What if patches for interest points look similar => Increase Patch size; What if same feature looks different due to scale, rotation, exposure… => Find Invariant Descriptor. Ex: **Harris Corner Detector**: Shift the window in any direction should give a large change in intensity (Flat region: no changes in all directions, Edge: no change along the edge direction, Corner: significant change in all directions). $E(u,v) = \sum_{x,y} w(x,y)[I(x+u, y_v) - I(x,y)]^2$, where $(u,v)$ is the shift direction and $w(x,y)$ is the window function (can use Step function or Gaussian). If $(u,v)$ is small: $E(u,v) \approxeq [u,v]\,M\begin{bmatrix} u \\ v \end{bmatrix}$, where $M = \sum_{x,y} w(x,y)\begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix}$. $I_x = \frac{\partial I}{\partial x}$, $I_y = \frac{\partial I}{\partial y}$, $I_xI_y = \frac{\partial I}{\partial x}\frac{\partial I}{\partial y}$, and $M \in R^{2\times 2}$ => Algorithm: 1. Computer $M$. 2. Compute the response function $R = \det M - k(\text{trace } M)^2$, where $k$ is a constant (usually within 0.04~0.06) 3. Find points with large corner response ($R > \theta$) 4. Take the points of local maxima of $R$. The corner response $R$ is invariant to image shift and rotation, but only partially invariant to intensity scale ($I \to \alpha I$). However, it is non-invariant to image scale (window size changes). To improve to become **scale invariant** => **Naive approach:** change the window size, find the one with the highest response.

**Feature Selection:** 1, NMS: Choose the maximum response within the window 2. Adaptive NMS: We want fixed number of features per image and should be evenly distributed: the radius taken is the distance till the nearest interest point of greater corner score. Discard all other points within the adaptive radius.

Feature Descriptor should be **Invariant** and **Distinctive**. We can also use Multi-Scale Oriented Patches (MOPS): Find local orientation (dominant direction of the local gradients) and extract the image patch.

**Feature Matching:** Given a feature in $I_1$, define distant function (ex: SSD) between two feature descriptors and test all features in $I_2$ with min distance. What if have ambiguous matches => The first matches' distance (cost) should be smaller than the second smallest one by a margin.

**Removing Outliers:** Can use RANSAC (Random Sample Consensus) => 1. Select four feature pairs at random to compute the homography matrix $H$, 2. Compute inliers where $SSD(p'_i, Hp_i) < \varepsilon$ , 3. Keep the largest set of inliers, back to 1 if doesn't have maximum inliers, 4. Recompute $H$ based on all of the inliers.