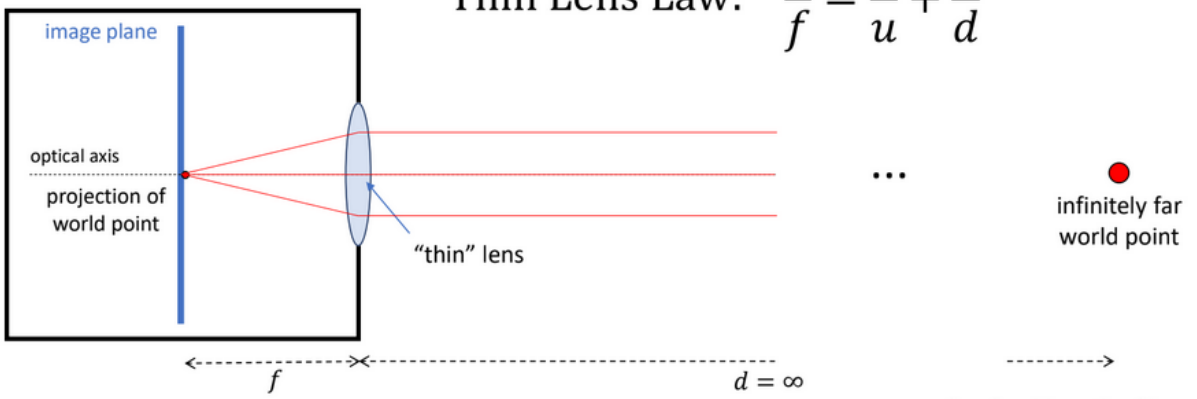
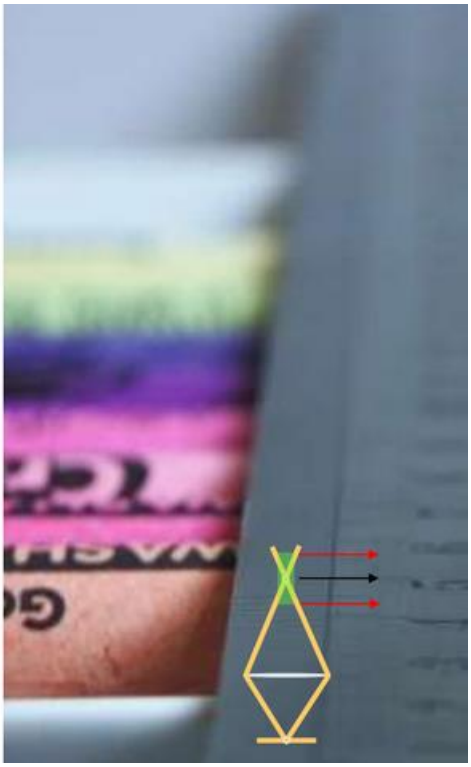
Thin Lens Law: $\frac{1}{f} = \frac{1}{u} + \frac{1}{d}$



f = focal length of lens
u = image distance
d = object distance

Depth of Field

- DoF = range of distances where blur < 1 sensor pixel!
- Things that affect DoF:
  - pixel size
  - aperture
  - lens focal length
- Cellphone camera:
  - wide-angle lens (short focal length)
  - need to fake DoF! (portrait mode)



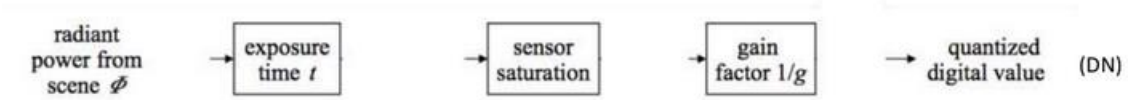
Aperture

- Expressed as $f/<\text{value}>$ (f-stop)
- e.g. this lens is 50mm and f/1.8
- f/1.8 is maximum aperture
- ➔ $\max(D) = \frac{50}{1.8} \approx 27.8$ mm

- $\uparrow D \downarrow \Delta t \Rightarrow$ small DoF
- $\downarrow D \uparrow \Delta t \Rightarrow$ large DoF

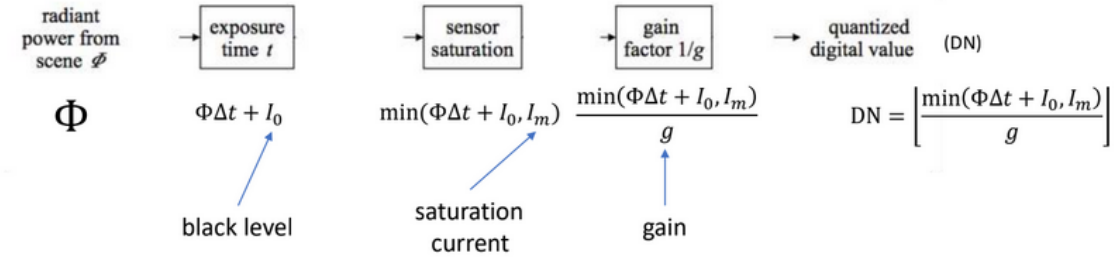
$\uparrow D \downarrow \Delta t$
(large aperture,
short exposure)

- High ISO ➔ brighter image
- High ISO ➔ higher noise



$$\Phi = \iint_{\overline{q}\,\lambda} H(\overline{q},\lambda)\, S(\overline{q})\, Q(\lambda)\, d\overline{q}\, d\lambda$$

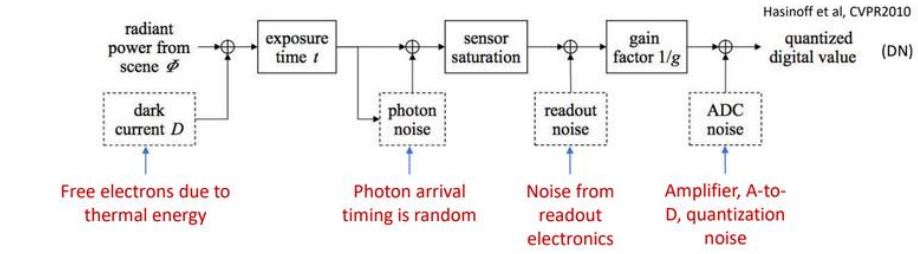
- spatial response at collection site (unitless): $S(\overline{q})$
- incident spectral irradiance (photons/s at given wavelength): $H(\overline{q},\lambda)$
- quantum device efficiency (electrons collected per incident photon at given wavelength): $Q(\lambda)$
- pixel footprint: $\overline{q}$
- wavelength: λ



gamma correction: $$f(DN) = f\left(\left\lfloor \frac{\min(\Phi\Delta t + I_0, I_m)}{g} \right\rfloor\right)$$

camera response function $$f_{\text{camera}}(\Phi\Delta t) = f_\gamma\,(DN(\Phi\Delta t)) = f_\gamma\left(\left\lfloor \frac{\min(\Phi\Delta t + I_0, I_m)}{g} \right\rfloor\right)$$

Sources of Noise: Putting them all Together



- $\text{Poisson}(\lambda = \Phi\Delta t)$ — Large $\Phi\Delta t$: $\approx \text{Normal}(\mu = \lambda, \sigma = \sqrt{\lambda})$
- $\text{Poisson}(\lambda = D\Delta t)$ — Large $D\Delta t$: $\approx \text{Normal}(\mu = \lambda, \sigma = \sqrt{\lambda})$
- $\text{Normal}(\mu = 0, \sigma = \sigma_r)$
- $\text{Normal}(\mu = 0, \sigma = \sigma_{ADC})$

mean(e^-)= $\min\{I_0 + \phi\Delta t + D\Delta t, I_m\}$
variance(e^-)= $\phi\Delta t + D\Delta t + I_0 + \sigma_r^2 + \sigma_{ADC}^2 \cdot g^2$

mean(DN)= $\min\{\frac{I_0+\phi\Delta t+D\Delta t}{g}, \frac{I_m}{g}\}$
variance(DN)= $\frac{\phi\Delta t}{g^2} + \frac{D\Delta t}{g^2} + \frac{I_0+\sigma_r^2}{g^2} + \sigma_{ADC}^2$

Approach 2: Blue Screen Matting

- Assume background contains only blue, i.e. $B = [B_r = 0, B_g = 0, B_b]$
  - Note: allows lighting to vary, because blue brightness not fixed, i.e. could be [0, 0, 1] or [0, 0, 0.5]
  - Matting equations simplify:

$$C_r = \alpha_F F_r + (1 - \alpha_F)B_r$$
$$C_g = \alpha_F F_g + (1 - \alpha_F)B_g$$
$$C_b = \alpha_F F_b + (1 - \alpha_F)B_b$$

  - 3 equations, 3 unknowns, and can still use α

Approach 2: Downsides

- Having no blue in foreground in practice is difficult!
- Not good for people with blue eyes (i.e. Hollywood movie actors/actresses)
- Excludes any color with blue component – i.e. 2/3 of all hues, including gray, pastels and white!
- ”Blue/green spilling”: light reflected off background onto foreground, making it have some blue component

Approach 3: Gray or Skin Colored Foreground

- Still blue screen, i.e. $B = [B_r = 0, B_g = 0, B_b]$
- Assume foreground colour is either:
  - Gray: i.e. $F_r = F_g = F_b$
  - "Flesh": $F = [F_r = d, F_g = \frac{d}{2}, F_b = \frac{d}{2}]$ where d depends on skin tone
- Equations simplify (e.g. gray):

$$C_r = \alpha_F F + (1 - \alpha_F) B_r$$
$$C_g = \alpha_F F + (1 - \alpha_F) B_g$$
$$C_b = \alpha_F F + (1 - \alpha_F) B_b$$

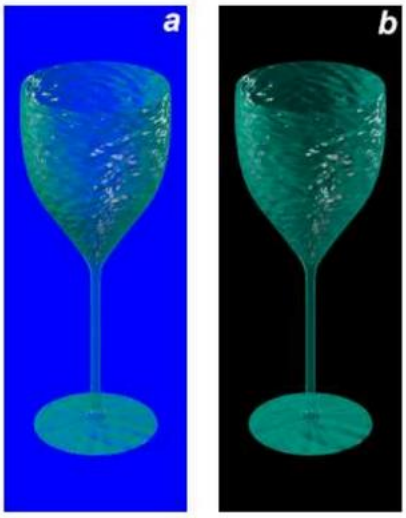
- 2 unknowns, 3 equations

Approach 4: Triangulation Matting

- Increase number of equations, by using two different backgrounds
- Assume backgrounds $B_0 = [B_{0,r}, B_{0,g}, B_{0,b}]$ and $B_1 = [B_{1,r}, B_{1,g}, B_{1,b}]$ known

$$C_{0,r} = \alpha_F F_r + (1 - \alpha_F) B_{0,r}$$
$$C_{0,g} = \alpha_F F_g + (1 - \alpha_F) B_{0,g}$$
$$C_{0,b} = \alpha_F F_b + (1 - \alpha_F) B_{0,b}$$
$$C_{1,r} = \alpha_F F_r + (1 - \alpha_F) B_{1,r}$$
$$C_{1,g} = \alpha_F F_g + (1 - \alpha_F) B_{1,g}$$
$$C_{1,b} = \alpha_F F_b + (1 - \alpha_F) B_{1,b}$$

- 6 equations, 4 unknowns



$$C_r = F_r' + (1-\alpha_F)B_r$$
$$C_g = F_g' + (1-\alpha_F)B_g$$
$$C_b = F_b' + (1-\alpha_F)B_b$$

Linear

Define $C_\Delta \equiv C - B$

Then

$$C_{\Delta r} = F_r' - \alpha_F B_r$$
$$C_{\Delta g} = F_g' - \alpha_F B_g$$
$$C_{\Delta b} = F_b' - \alpha_F B_b$$

$$\Rightarrow \begin{bmatrix} C_r - B_r \\ C_g - B_g \\ C_b - B_b \end{bmatrix} = \begin{bmatrix} C_{\Delta r} \\ C_{\Delta g} \\ C_{\Delta b} \end{bmatrix} = \begin{bmatrix} 1 & & & -B_r \\ & 1 & & -B_g \\ & & 1 & -B_b \end{bmatrix} \begin{bmatrix} F_r' \\ F_g' \\ F_b' \\ \alpha_F \end{bmatrix} = \begin{bmatrix} 1 & & & -B_r \\ & 1 & & -B_g \\ & & 1 & -B_b \end{bmatrix} \begin{bmatrix} \alpha_F F_r \\ \alpha_F F_g \\ \alpha_F F_b \\ \alpha_F \end{bmatrix}$$

Triangulation Matting : In Matrix Form

Similarly, define $F' \equiv \alpha_F F$, and Define $C_\Delta \equiv C - B$

$$\begin{aligned} C_{0,r} &= \alpha_F F_r + (1-\alpha_F)B_{0,r} \\ C_{0,g} &= \alpha_F F_g + (1-\alpha_F)B_{0,g} \\ C_{0,b} &= \alpha_F F_b + (1-\alpha_F)B_{0,b} \\ C_{1,r} &= \alpha_F F_r + (1-\alpha_F)B_{1,r} \\ C_{1,g} &= \alpha_F F_g + (1-\alpha_F)B_{1,g} \\ C_{1,b} &= \alpha_F F_b + (1-\alpha_F)B_{1,b} \end{aligned} \quad \Rightarrow \quad \begin{bmatrix} C_{\Delta 0,r} \\ C_{\Delta 0,g} \\ C_{\Delta 0,b} \\ C_{\Delta 1,r} \\ C_{\Delta 1,g} \\ C_{\Delta 1,b} \end{bmatrix} = \begin{bmatrix} 1 & & & -B_{0,r} \\ & 1 & & -B_{0,g} \\ & & 1 & -B_{0,b} \\ 1 & & & -B_{1,r} \\ & 1 & & -B_{1,g} \\ & & 1 & -B_{1,b} \end{bmatrix} \begin{bmatrix} F_r' \\ F_g' \\ F_b' \\ \alpha_F \end{bmatrix}$$

Pseudo-inverse to compute the solution

Inverse Camera Response Function

- Recall C.R.F. gives the pixel value (Z) for a given scene irradiance:

$$Z = f_{\text{camera}}(\Phi \Delta t)$$

- We want the the scene irradiance given the pixel value (Z), i.e. the inverse C.R.F.:

$$\Phi \Delta t = {f_{\text{camera}}}^{-1}(Z)$$

photons/sec

exposure time

pixel value

Merging Multiple Exposures: Debevec et al.

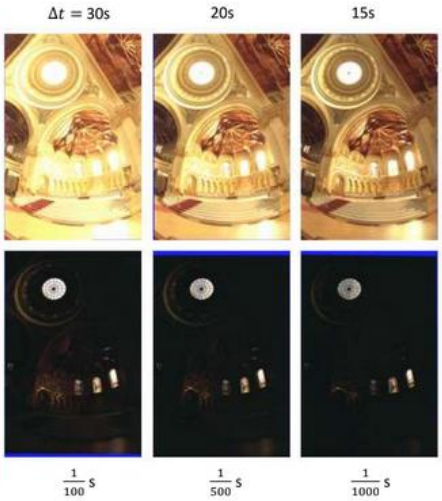
- Algorithm Outline: (see Debevec et al. in readings)

  for each pixel location i and pixel value Z_i:
  for each photo $j = 1, \ldots, P$ with exposure time Δt_j:
  estimate Φ_{ij} at pixel location i given Δt_j, Z_i

  combine estimates $\Phi_{ij}, where\ j = 1, \ldots, P$ to get Φ_i

  Result: HDR image where each pixel is a float based on our estimate of the scene irradiance Φ_i for each pixel

- i is the pixel location, regarding image as flattened 1D array of pixels



Finally: Merging Multiple Exposures

- Algorithm HDR: (see Debevec et al. in readings)
  for each pixel location i and pixel value Z_i:

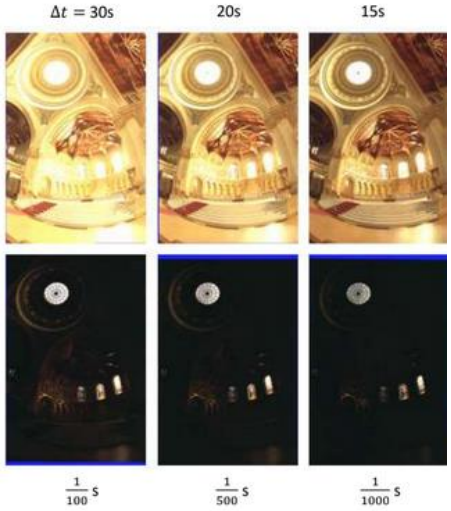
  for each photo $j = 1, \ldots, P$:

$$\log \Phi_{ij} = g(Z_{ij}) - \log \Delta t_j$$

$$\log \Phi_i = \frac{\sum_{j=1}^{P} w(Z_{ij}) \log \Phi_{ij}}{\sum_{j=1}^{P} w(Z_{ij})}$$

  Where $w(Z_{ij})$ is a weighting factor that depends on the pixel value. Pixels close to saturation (255), or close to the black level (0) are weighted lower.

  (Note: in the paper $E \equiv \Phi$)



Estimating Derivatives For Image Row r

"Sliding window" algorithm:

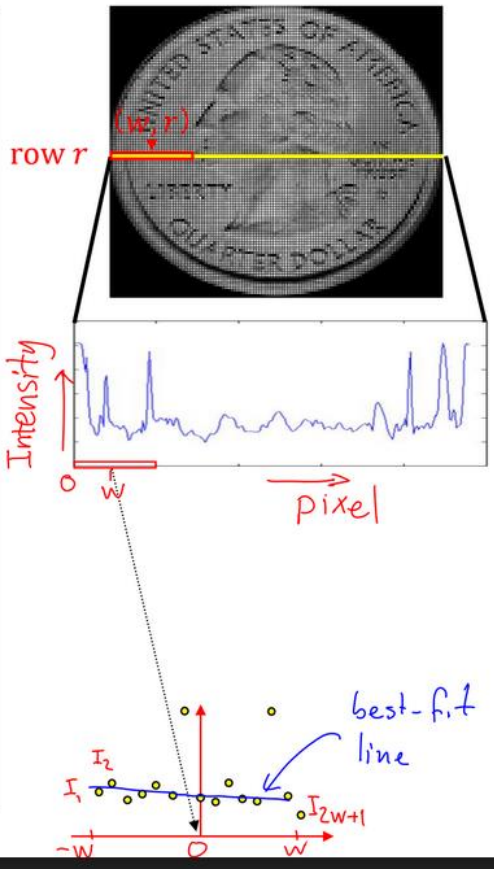
- Define a "pixel window" centered at pixel (w,r)

  In this demonstration, the pixel index starts from 0, so the patch is (0,r) to (2w,r), centering at w.
- Fit n-degree poly to window's intensities (usually n=1 or 2)
- Assign the poly's derivatives at x=0 to pixel at window's center

$$\frac{dI}{dx}(w) \longleftrightarrow \frac{dI}{dx}(0)$$

  image coordinate ↔ patch coordinate

- "Slide" window one pixel over, so that it is centered at pixel (w+1,r)
- Repeat 1-4 until window reaches right image border



RANSAC Algorithm

Given:

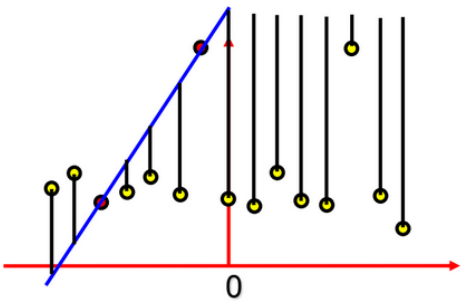
- n = degree of poly
- p = fraction of inliers
- t = fit threshold
- p_s = success probability

Repeat at most K times:

1. Randomly choose n+1 pixels
2. Fit n-degree poly
3. Count pixels whose vertical distance from poly is < t
4. If there are at least (2w+1)p pixels, EXIT LOOP
   a. Label them as inliers
   b. Fit n-degree poly to all inlier pixels

Q: What should K be?

- Probability we chose an inlier pixel: p
- Probability we chose (n+1) inlier pixels: p^{n+1}
- Prob at least 1 outlier chosen: $1-p^{n+1}$
- Prob at least 1 outlier chosen in all K trials: $(1-p^{n+1})^K$



Cross-Correlation vs. Convolution (discrete)

Cross-correlation (sliding inner-product)

$\forall r, c:$

$$\mathrm{CC}(I,T,r,c) = \sum_{a=-w}^{w}\sum_{b=-w}^{w} I(r+a,c+b)\,T(a,b)$$

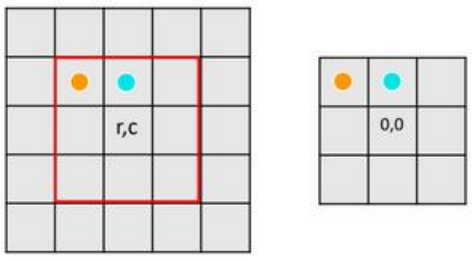


Image Template

● × ● + ● × ● +...

Lots of useful properties.

Convolution

$\forall r, c:$

$$I * T\,(r,c) = \sum_{a=-w}^{w}\sum_{b=-w}^{w} I(r-a,c-b)\,T(a,b)$$

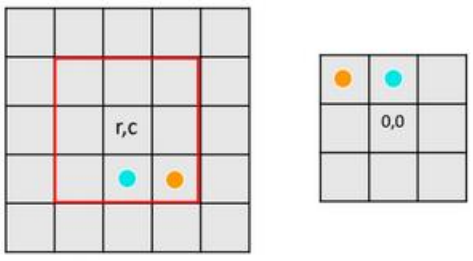
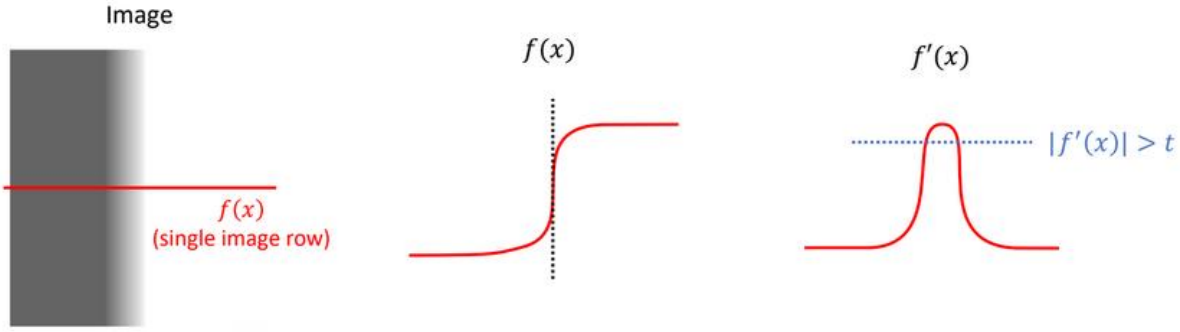


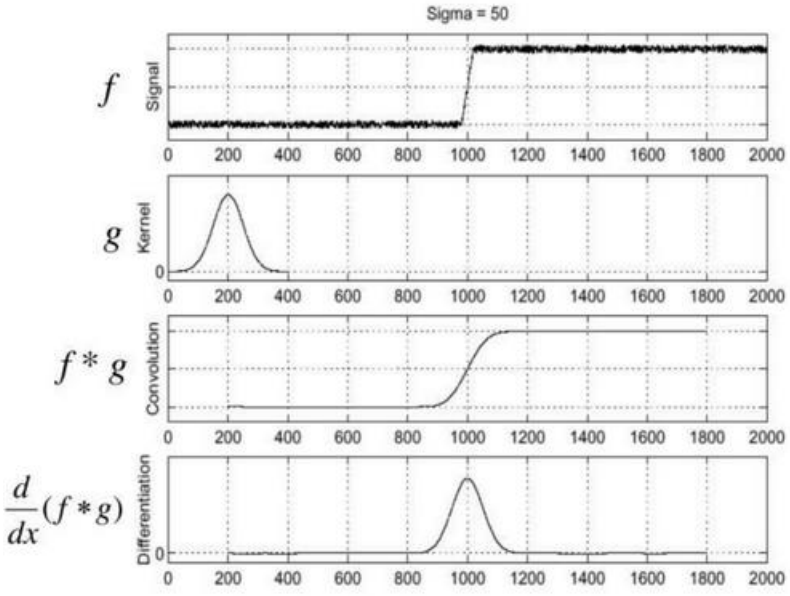
Image Template

● × ● + ● × ● +...

Issues with Gradient Edges

- Using gradient filters to find edges is not ideal:
  - Edge from gradient magnitude is “thick”, ideally as localized to one pixel
  - This is because we threshold the gradient magnitude
  - Can possibly include several pixels around the true edge location





Sobel Filter for Edges

$$Sobel_x = \frac{1}{8}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$Sobel_y = \frac{1}{8}\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

- Sobel filter is **the** standard image gradient filter
- There are many others however!
- Typically the normalization (1/8) term is ignored

Final

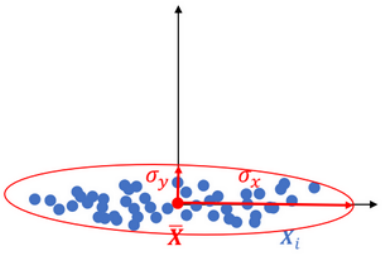
Covariance

- We define covariance:

$$\text{cov}(X,Y) = \frac{1}{N}\sum (X_i - \bar{X}^X)(Y_i - \bar{X}^Y)$$

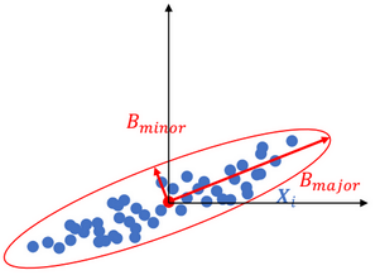
- Notice, variance is a special case of covariance

$$\sigma^2(X) = \text{cov}(X,X)$$



- In 2D, we have 4 possible covariances, represented in the **covariance matrix**

$$\Sigma(X,Y) = \begin{bmatrix} \text{cov}(X,X) & \text{cov}(X,Y) \\ \text{cov}(Y,X) & \text{cov}(Y,Y) \end{bmatrix}$$



Principle Component Analysis (PCA) Algorithm

Given N image patches of M dimensions:

1) Calculate mean of image patch vectors

$$\bar{\boldsymbol{X}} = \frac{1}{N}\sum \boldsymbol{X}_i$$

2) Subtract the mean from all patches (centre)

$$\boldsymbol{Z}_i = \boldsymbol{X}_i - \bar{\boldsymbol{X}}$$

3) Create an $M \times N$ matrix of all centred patch vectors (arranged as columns of matrix)

$$Z = [\boldsymbol{Z}_1 \quad \boldsymbol{Z}_2 \cdots \quad \boldsymbol{Z}_N]$$

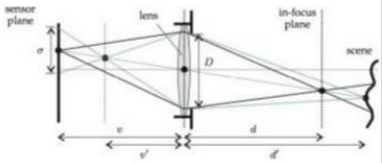
4) Find ***eigenvectors*** $\boldsymbol{B}_1, \ldots, \boldsymbol{B}_d$ corresponding to the d (where $d \ll M$) largest ***eigenvalues*** $\lambda_1, \ldots, \lambda_d$ of the **covariance matrix**

$$\Sigma = ZZ^T$$

Convolution in imaging

Motion blur

- Blur kernel shape ≈ path of motion



Out of focus blur

- Blur kernel is different for objects at different distance

Aperture shape

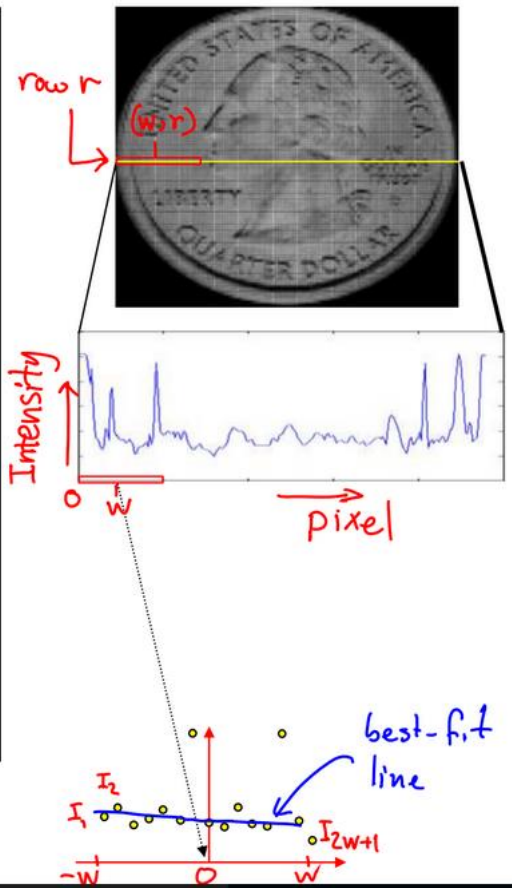
Solar eclipse 2017 by Yani

Convolution result of the eclipse (signal) and holes formed by leaves (kernel)

Template Matching (1D)

"Sliding window" algorithm for template matching with template T

- Define a "pixel window" centered at pixel (w,r)
- Compute cross-correlation of T with patch centered at (w,r)
- "Slide" window one pixel over, so that it is centered at pixel (w+1,r)
- Repeat 1-4 until window reaches right image border



Difference of two Gaussian-smoothed versions of I:

$I * G_{\sigma_1} - I * G_{\sigma_2} = I * (G_{\sigma_1} - G_{\sigma_2})$

the DoG filter (just the difference between two Gaussian masks)

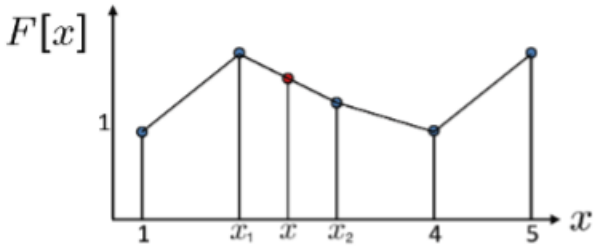
The Gaussian Pyramid

Goal: Develop representation to decompose images into information at multiple scales, to extract feature or structure of interest, to attenuate noise.

Applications:

- Efficient image coding
- Progressive transmission
- Image blending
- Image enhancement
- Efficient Processing

Linear Interpolation



d = 1 in this example

- Linear interpolation from our discretized F:

$$G(x) = \frac{x_2 - x}{x_2 - x_1} F(x_1) + \frac{x - x_1}{x_2 - x_1} F(x_2)$$

Smoothing Filter in 1D: Derivation from 4 Criteria

1. $\hat{w}$ always has 5 elements (aka "5-tap" filter)

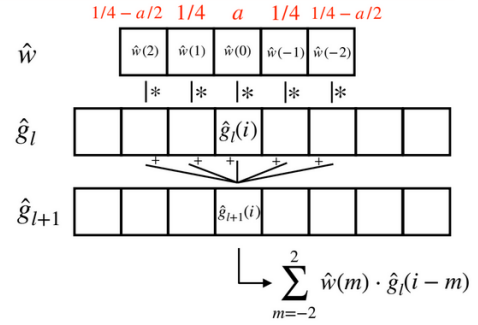
2. $\hat{w}$ symmetric about o:

$$\hat{w} = \begin{bmatrix} c \\ b \\ a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 1/4 - a/2 \\ 1/4 \\ a \\ 1/4 \\ 1/4 - a/2 \end{bmatrix}$$

3. applying $\hat{w}$ to a **constant** image does not change it

$$\sum_{m=-2}^{2} \hat{w}(m) = 1$$

$$\iff a + 2b + 2c = 1$$



4. Equal contribution

$$a + 2c = 2b = 1/2$$

To satisfy Criteria 1-4 we have a 2 equations & 3 unknowns

⇒ a remains a free parameter

$\hat{w}(2) = \hat{w}(-2) = \frac{1}{4} - \frac{a}{2}, \hat{w}(-1) = \hat{w}(1) = \frac{1}{4}$

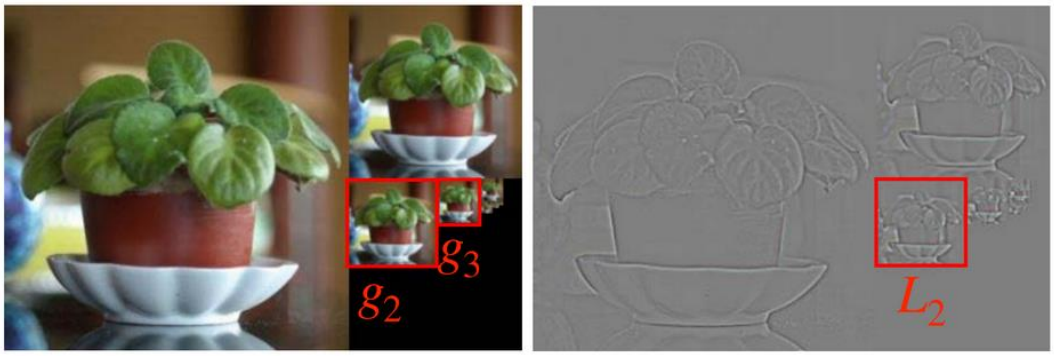
usually $a \in [0.3, 0.6]$

$$g_{l+1} = \text{REDUCE}(g_l)$$

$$g_{l+1}(i,j) = \sum_{m=-2}^{2} \sum_{n=-2}^{2} w(m,n) \cdot g_l(2i - m, 2j - n)$$

$$\text{EXPAND}(g_l) = 4 \sum_{m=-2}^{2} \sum_{n=-2}^{2} w(m,n) \cdot g_l\left(\frac{i-m}{2}, \frac{j-n}{2}\right)$$

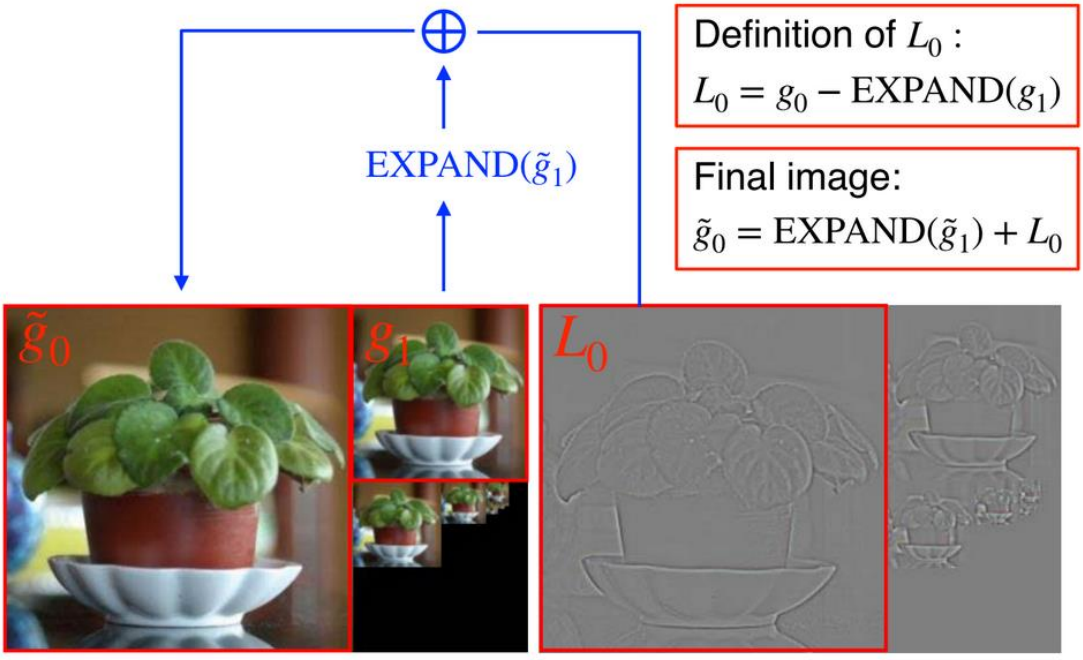
$$L_2 = g_2 - \text{EXPAND}(g_3)$$



Pyramid-Based Image Editing

1. Edit g_1 to obtain $\tilde{g}_1$.

2. Add the details using L_0

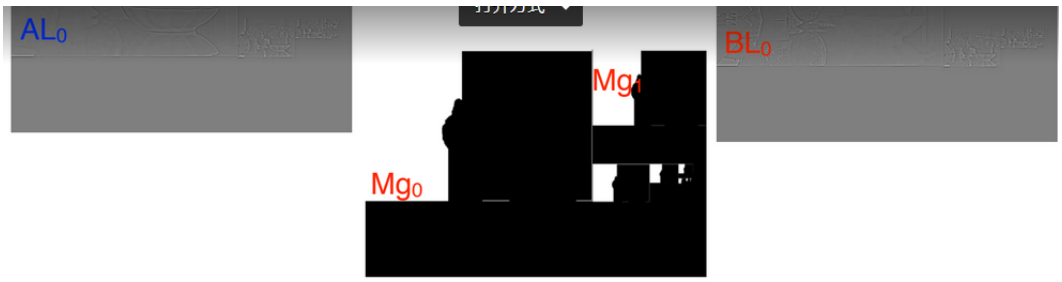


Definition of L_0 :

$L_0 = g_0 - \text{EXPAND}(g_1)$

Final image:

$\tilde{g}_0 = \text{EXPAND}(\tilde{g}_1) + L_0$



1. Compute A's Laplacian pyramid: $AL_0,\ldots, AL_{N-1}, Ag_N$

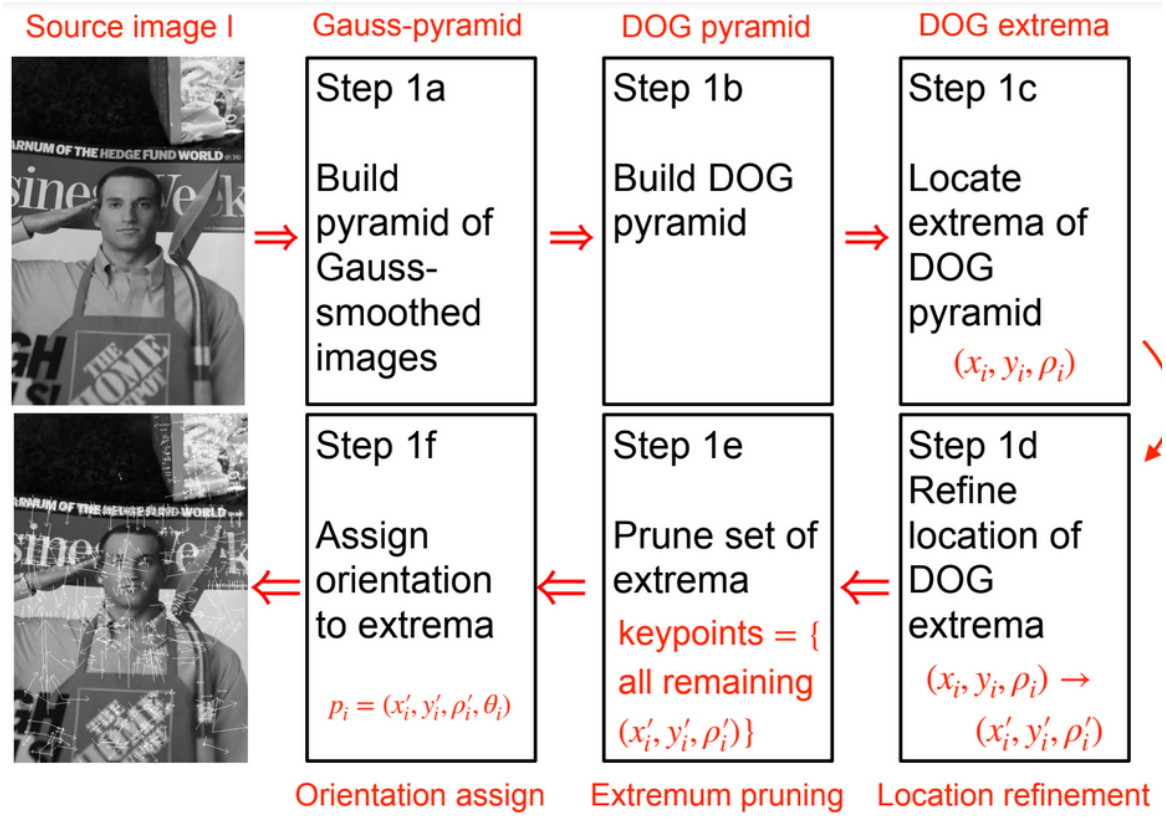
3. Compute M's Gaussian pyramid: $Mg_0,\ldots, Mg_N$

2. Compute B's Laplacian pyramid: $BL_0,\ldots, BL_{N-1}, Bg_N$

Pyramid Blending Algorithm

4. Compute the Laplacian pyramid, $SL_0, SL_1,\ldots SL_{N-1}, Sg_N$ by applying the matting equation with matte Mg_l.

$$SL_l(i,j) = Mg_l(i,j)AL_l(i,j) + (1 - Mg_l(i,j))BL_l(i,j)$$



Condition for detecting a "strong" extremum (x_i', y_i', ρ_i')

$|D(x_i', y_i', \rho_i')| =$ large

in practice, > 0.03

assumes image I has pixel intensities in the range [0, 1]

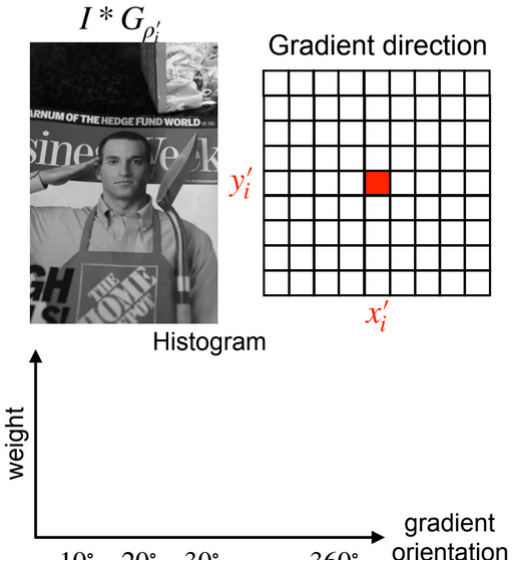
$D(x, y, k^S\sigma)$

Compute hessian H of

$D(x, y, \rho_i')$ at $(x, y) = (x_i', y_i')$

Prune if $\frac{\mathrm{Tr}^2(H)}{\mathrm{Det}(H)} > \left(\frac{11}{10}\right)^2$ 11^2/10

Assigning an orientation θ_i to keypoint (x_i', y_i', ρ_i') :



A. Compute smoothed image
B. Compute gradient magnitude & orientation in neighbourhood of (x_i', y_i') in $I * G_{\rho_i'}$
C. Compute histogram of orientations
D. Assigned orientation θ_i = highest peak in histogram

3. Match f_i

a. Compute $\|f_i - f_j'\|$ for all j

b. Compute fraction $\phi = \frac{\|f_i - f_{j^*}'\|}{\|f_i - f_{j^{**}}'\|}$ where f_{j^*}' is the closest descriptor in I' and $f_{j^{**}}'$ is 2nd-closest.

c. Match f_i to f_{j^*}' if $\phi < 0.8$

$ax + by + c = 0$

(line, parameters)

- In homogeneous coordinates

$$\begin{bmatrix} a & b & c \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = 0$$

or $\ell^T \cdot p = 0$

(ℓ: vector holding line parameters; p: vector holding homogeneous coordinates of a point)

Line through 2 points

$$\ell = p_1 \times p_2 = \begin{bmatrix} 0 & -p_1^z & p_1^y \\ p_1^z & 0 & -p_1^x \\ -p_1^y & p_1^x & 0 \end{bmatrix} \begin{bmatrix} p_2^x \\ p_2^y \\ p_2^z \end{bmatrix}$$

Intersection of 2 lines

$$p = \ell_1 \times \ell_2 = \begin{bmatrix} 0 & -\ell_1^z & \ell_1^y \\ \ell_1^z & 0 & -\ell_1^x \\ -\ell_1^y & \ell_1^x & 0 \end{bmatrix} \begin{bmatrix} \ell_2^x \\ \ell_2^y \\ \ell_2^z \end{bmatrix}$$

- Is $\begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} \cong \begin{bmatrix} -2 \\ 0 \\ 4 \end{bmatrix}$? no!

Definition:

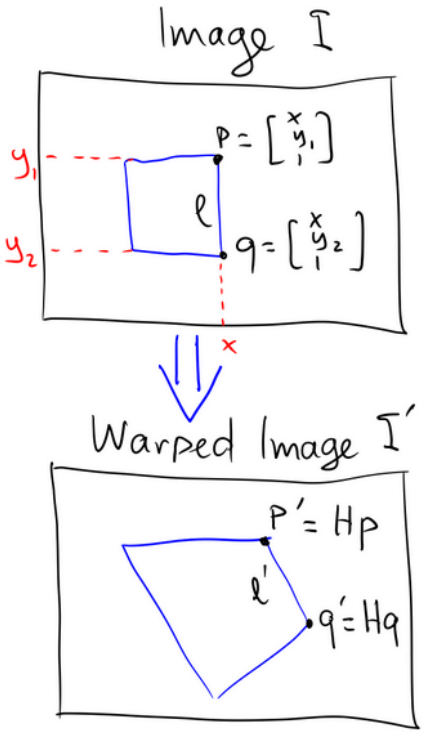
Homogeneous representation of p

> p represented by any 3D vector $\begin{bmatrix} \lambda x \\ \lambda y \\ \lambda \end{bmatrix}$ with $\lambda \neq 0$

- Homogeneous (a.k.a. Projective) representation of p

> image coordinates $\begin{bmatrix} x \\ y \end{bmatrix} \longrightarrow$ homogeneous 2D coordinates $\begin{bmatrix} \lambda x \\ \lambda y \\ \lambda \cdot 1 \end{bmatrix}$ $\lambda \neq 0$

Warping Images Using a Homography



The matrix H is called a Homography

- Linear warping equation

$$I(p) = I'(Hp)$$

intensity at pixel in source image with homogeneous coordinates p

intensity at pixel in warped image with homogeneous coordinates $p' = Hp$

- Note: Scaling H by a factor $\lambda \neq 0$ does not change the homography:

$$(\lambda \cdot H)p = H \cdot (\lambda p) \cong Hp$$

- Computing warp I' from I and H

① Compute H^{-1}

② To compute color of pixel (u,v) in warped image:

- compute $\begin{bmatrix} a \\ b \\ c \end{bmatrix} = H^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$
- copy color from $I(a/c, b/c)$

a

H_2

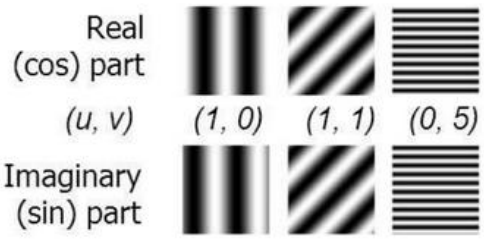
because some matches maybe incorrect (i.e. outliers) RANSAC is used

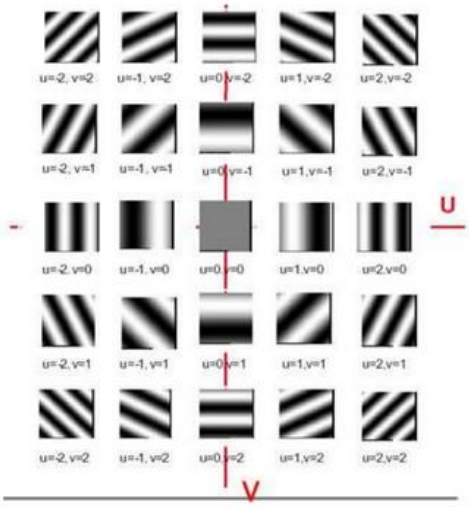
The Autostitch Algorithm

1. Run SIFT on each image
2. Match SIFT features across all photos
3. For every pair of photos:
   - Choose 4 matches & compute H
4. Warp photos according to H
5. Blend photos using pyramid blending

Fourier Transform as Basis

$$exp[j\,2\pi(ux + vy)] =$$
$$cos[2\pi(ux + vy)] + j\,sin[2\pi(ux + vy)]$$





$$F(\omega) = \int_{-\infty}^{\infty} f(x)\, e^{-i\omega x}\, dx$$

- $f(x)$ is the function we want to transform to the frequency domain
- $F(\omega)$ is the function in the frequency domain, where $\omega = 2\pi f$

The continuous 2D Fourier transform is defined:

$$F(u,v) = \iint_{-\infty}^{\infty} f(x,y)\, e^{-i(ux+vy)}\, dx\, dy$$

The discrete 2D Fourier transform is defined:

$$F(u,v) = \sum_{x} \sum_{y} f(x,y)\, e^{-i(ux+vy)}$$

Images are just a discrete 2D function, so we can also represent them in the frequency domain

- Let $\mathcal{F}\{f\}$ respresents the Fourier transform of f
- If $\mathcal{F}\{f\} = F$, then $\mathcal{F}^{-1}\{F\} = f$

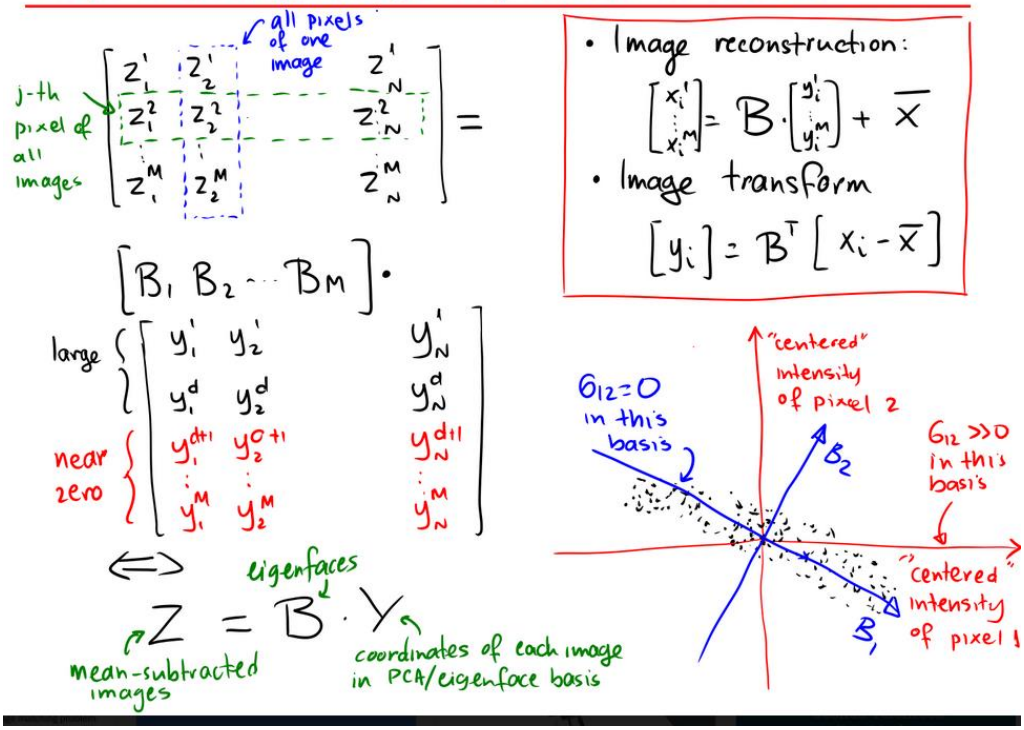
1. $\mathcal{F}\{f * g\} = \mathcal{F}\{f\} \cdot \mathcal{F}\{g\}$ or $f * g = \mathcal{F}^{-1}\left\{\mathcal{F}\{f\} \cdot \mathcal{F}\{g\}\right\}$
   F.T. of **Convolution** in the **spatial domain** is the same as **multiplication** in the **frequency domain**

2. $\mathcal{F}\{f \cdot g\} = \mathcal{F}\{f\} * \mathcal{F}\{g\}$ or $f \cdot g = \mathcal{F}^{-1}\{\mathcal{F}\{f\} * \mathcal{F}\{g\}\}$
   F.T. of **Multiplication** in the **spatial domain** is the same as **convolution** in the **frequency domain**

It turns out there is a much more efficient implementation of the Discrete Fourier Transform (DFT) called the Fast Fourier Transform (FFT)

For a 1D signal with N data points, DFT is $O(N^2)$, FFT is $O(N \log N)$

Representing Images by their PCA Basis



Wavelet vs. other transformations

Predefined basis

– similar to Fourier transform

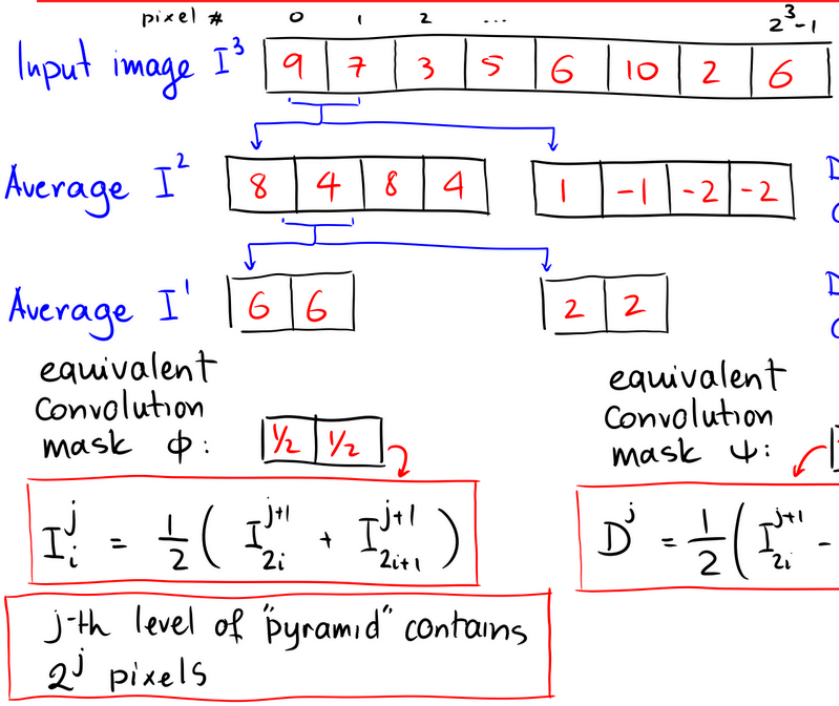
Reduce dimension (compression algorithm)

– similar to PCA

Transformation stores the pixel difference

– similar to Laplacian pyramid representation

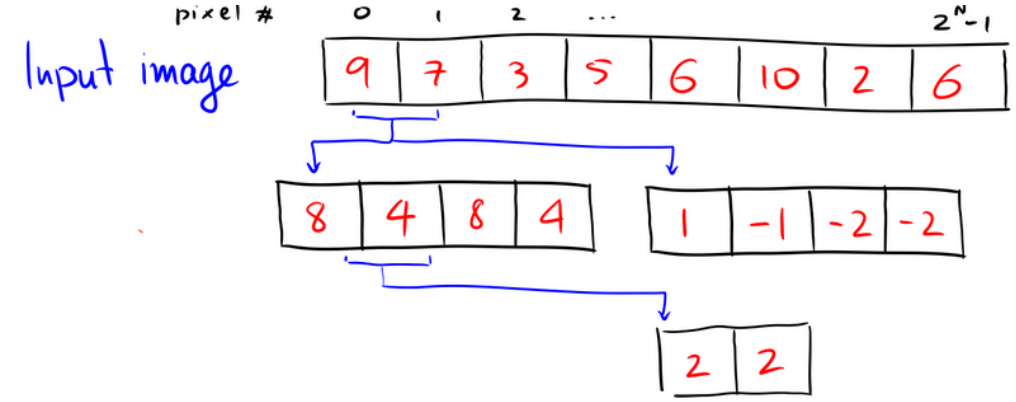
1D Haar Wavelet Transform: Recursive Definition



$$I_i^j = \frac{1}{2}\left(I_{2i}^{j+1} + I_{2i+1}^{j+1} \right)$$

$$D^j = \frac{1}{2}\left(I_{2i}^{j+1} - I_{2i+1}^{j+1} \right)$$

j-th level of "pyramid" contains 2^j pixels

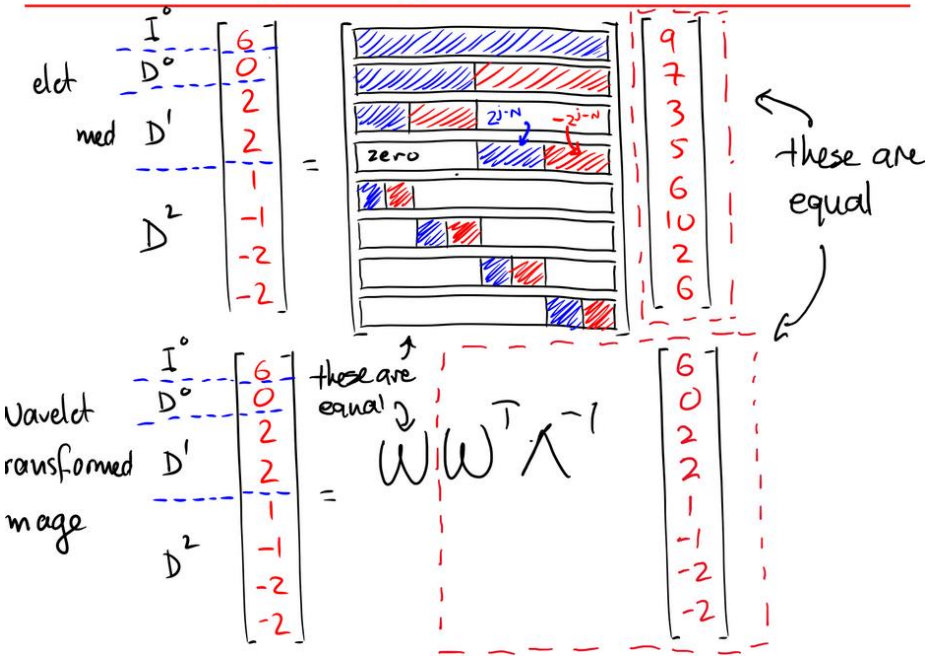


Wavelet transformed image

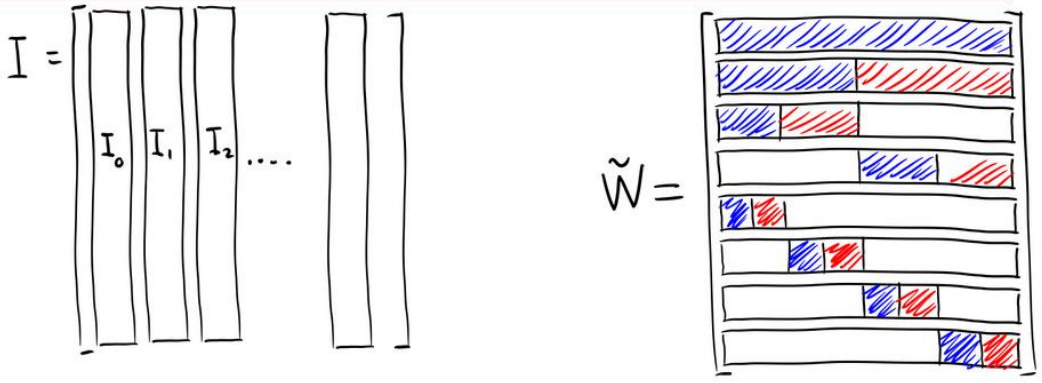
$$\begin{bmatrix} I^0 \\ D^0 \\ D^1 \\ D^2 \end{bmatrix} = \begin{bmatrix} 6 \\ 0 \\ 2 \\ 2 \\ 1 \\ -1 \\ -2 \\ -2 \end{bmatrix} = \begin{bmatrix} 1/8 & -1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1/8 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1/4 & 1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 \\ & 0 & 0 & 0 & 0 & 1 & 1 & -1 & -1 \\ & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} 9 \\ 7 \\ 3 \\ 5 \\ 6 \\ 10 \\ 2 \\ 6 \end{bmatrix}$$

Original image

Reconstructing an Image from its Wavelet Coefs



The 2D Haar Wavelet Transform



To compute the wavelet transform of a 2D image:

1. Compute the 1D transform for each column and place the vectors $\tilde{W}I_i$ in a new image I'
2. Compute the 1D transform of each row of I'

Cross-dissolve

A weighted combination of two images, pixel-by-pixel

Image 0

Image 1

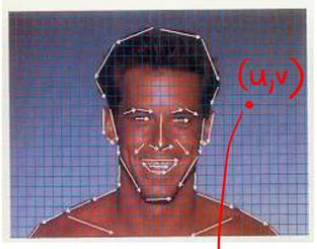
Combination controlled by a single interpolation parameter t:

Dest = t·(Image1) + (1-t)(Image2)

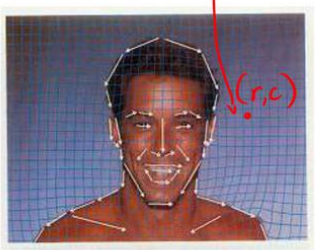
Morphing by Backward Mapping

To completely determine the morph we need to define the functions U(r,c), V(r,c)

Image 0



Intermediate warp t



Called "backward" because the loop is over the destination pixels

Backward mapping:

```
for r = rmin to rmax
  for c = cmin to cmax
    r' = U(r,c)
    c' = V(r,c)
    copy pixel at source (r',c')
      to destination (r,c)
```

Image 0

intermediate image

(U, V) are the mapping functions

Beier-Neely Field Warping Algorithm

For each pixel (r,c) in destination image

DSUM=(0,0)

weightsum = 0

for each line (P_i,Q_i)

calculate (u_i,v_i)

based on P_i,Q_i

calculate (r_i',c_i')

based on u,v & P_i',Q_i'

calculate displacement

$D_i=X_i'-X_i$ for this line

calculate weight

for line (P_i,Q_i)

DSUM += Di*weight

weightsum += weight

(r',c') = (r,c) + DSUM/weightsum

color at destination pixel (r,c) = color at source pixel (r',c')