Transformation of the **spatial** variables

Types of processing seen so far…

- Manipulate the dependent (pixel value) variable
  - range of the function

$$I_{\text{out}}(m,n) \rightarrow I_{\text{in}}(m,n)$$

  - pixel values at each location (m,n) have been altered according to its local or global properties of I_{old}

Next

- Manipulate the independent (spatial) variables
  - domain of the function

$$I_{\text{out}}(m,n) \rightarrow I_{\text{in}}(i,j)$$

Operating on the spatial variables

There are 2 possibilites:

1. Scaling the spatial variables (resampling)

$$f[m,n] \rightarrow g[i,j] = f[an,bn]$$

   – the size of the input and output images are different
   – $0>a=b>1$ ➔ downsampling and $0<a=b<1$ ➔ upsampling

2. Transforming the spatial variables (geometrical transformation)

$$f[m,n] \rightarrow g[i,j] = f[T(m,n)]$$

   – the sizes of the input and output images are equal
   – 1 is the same as 2 with T = [a,b]

Check: what is the difference between resizing and resampling?

1. Resampling

Scaling the spatial variable

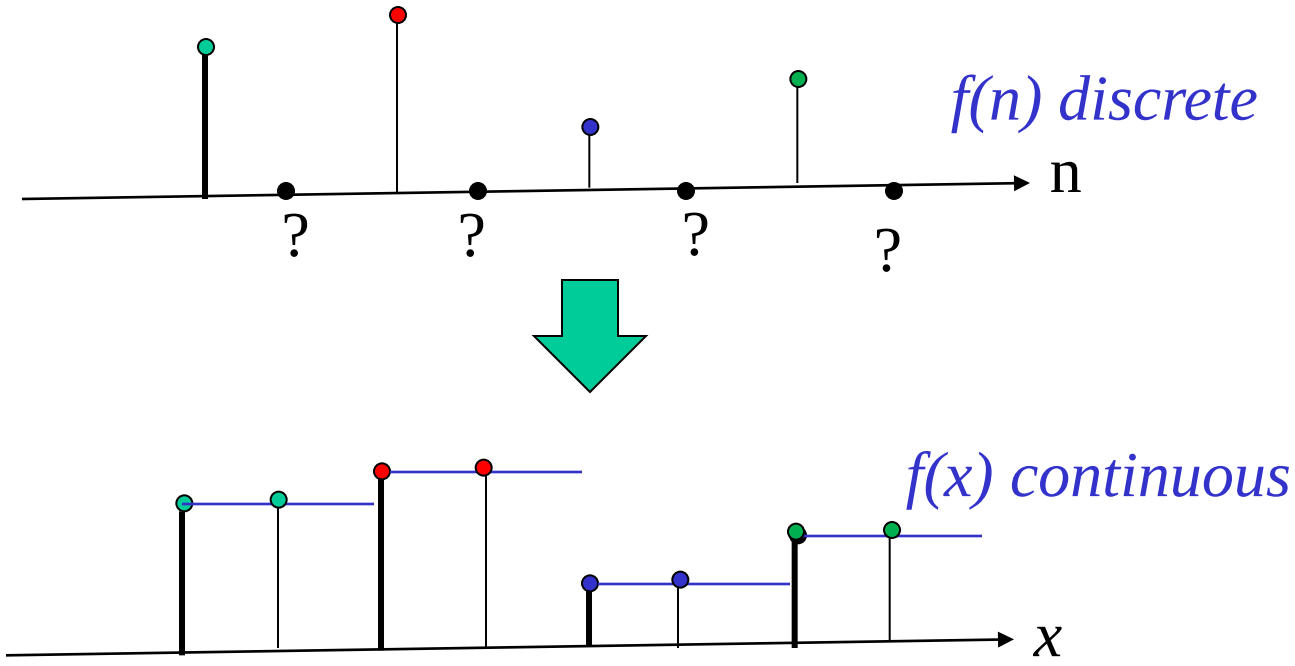
Consider a sequence *f[n]* with 4 samples

- How to double the length of this sequence? *y[n]=f[n/2]*
  upsampling

- How to halve the length of this sequence? *y[n]=f[2n]*
  downsampling

Upsampling – PC assumption

Task: doubling the length of a sequence
i.e. given 4 samples, find 4 more samples

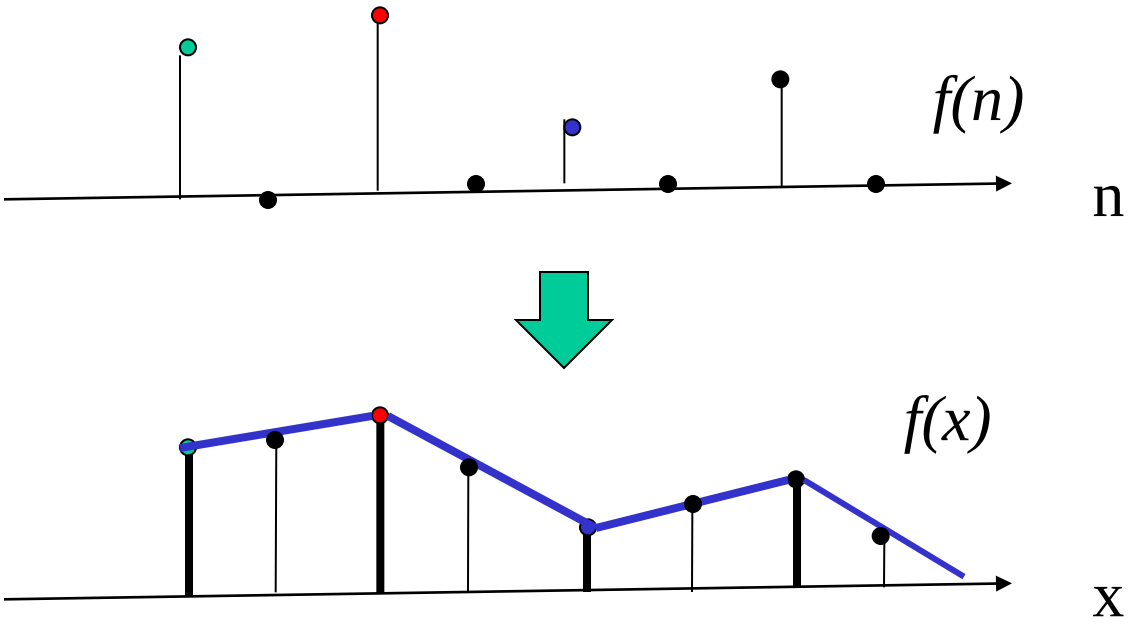


The function is assumed to be piecewise constant (PC)
- this is equivalent to replicating the previous pixel value

Upsampling – PL assumption

PC assumption is too simplistic.

Interpolation can be improved by assuming the function to be piecewise linear (PL)



This is linear interpolation

Upsampling in 2D (magnifying an image)

Magnification

Given $f[m,n]$ find $g[m,n]=f[\alpha m,\alpha n]$, $\alpha<1$

- g is larger than f in size
- also called *zooming in*

Approach: Upsample via interpolation

1. Reconstruct the continuous function $f(x,y)$
   - i.e. fit a function to describe the behaviour between samples of $f[m,n]$
   - simple to complex behaviour can be assumed between samples
2. Resample $f(x,y)$ to obtain g = $f[\alpha m,\alpha n]$

Note: In practice, $f(x,y)$ is not reconstructed fully. Only the estimates at the desired 'new' locations are estimated.

Magnifying an image

1. Given $I[m,n]$ create $I_M = I[am,an]$, $a<0$
2. Set all the 'new' pixels to 0
3. Find their value using *replication* (4a or 4b)

4a. *Replication by copying*

- Copy the value of known 'previous' pixel value to the 'new' pixel locations

4b. *Replication by convolution*

- Interpolate to find the 'new' pixel values by convolving I_M with h (an axa mask) red box is the mask origin

Ex. For $a = 2$, $h = \begin{bmatrix} 1 & 1 \\ 1 & \boxed{1} \end{bmatrix}$

Magnifying by pixel replication – example

10	**25**	**40**
25	**25**	**25**
50	**25**	**10**

Given image

10	10	25	25	40	40
10	10	25	25	40	40
25	25	25	25	25	25
25	25	25	25	25	25
50	50	25	25	10	10
50	50	25	25	10	10

After 2x2 magnification

Magnifying by bilinear interpolation

- Assume a piecewise linear variation between a pair of pixels and find the new values row- then column-wise
  - Fit a straight line between a pair of pixels
  - First order hold operation

- Interpolation can also be done as a convolution operation

Ex: Bilinear interpolation

$$h = \begin{bmatrix} 1/4 & 1/2 & 1/4 \\ 1/2 & \boxed{1} & 1/2 \\ 1/4 & 1/2 & 1/4 \end{bmatrix}$$

Achieves a 2x2 magnification

- Higher order interpolation: Cubic spline interpolation

Zooming – bilinear interpolation

10	**25**	**40**
25	**25**	**25**
50	**25**	**10**

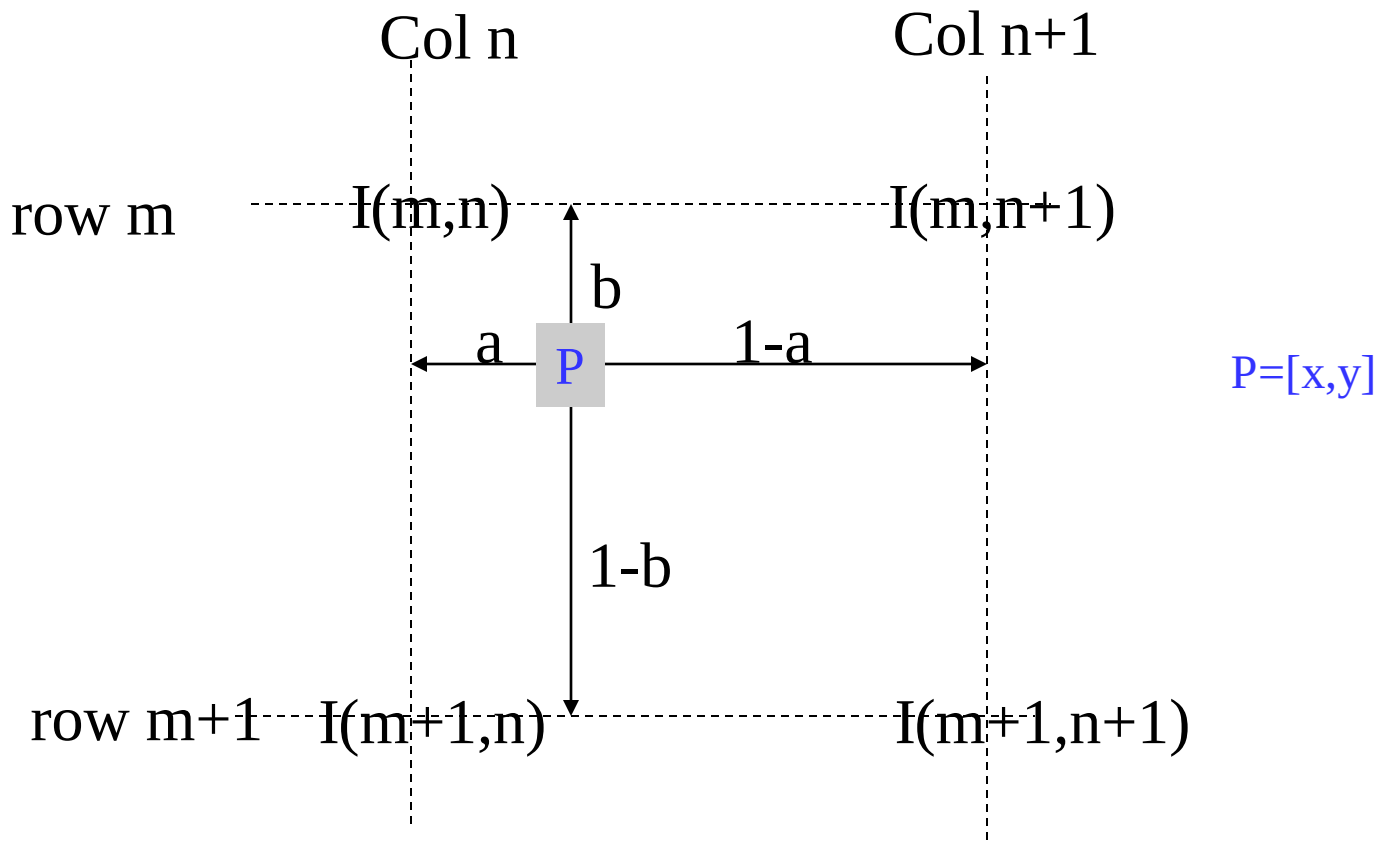
Bilinear →

10	17	25	32	40	20
17	21	25	28	32	16
25	25	25	25	25	12
37	28	25	21	17	8
50	32	25	17	10	5
25	16	12	8	5	2

PC ↓

10	10	25	25	40	40
10	10	25	25	40	40
25	25	25	25	25	25
25	25	25	25	25	25
50	50	25	25	10	10
50	50	25	25	10	10

General case of linear interpolation

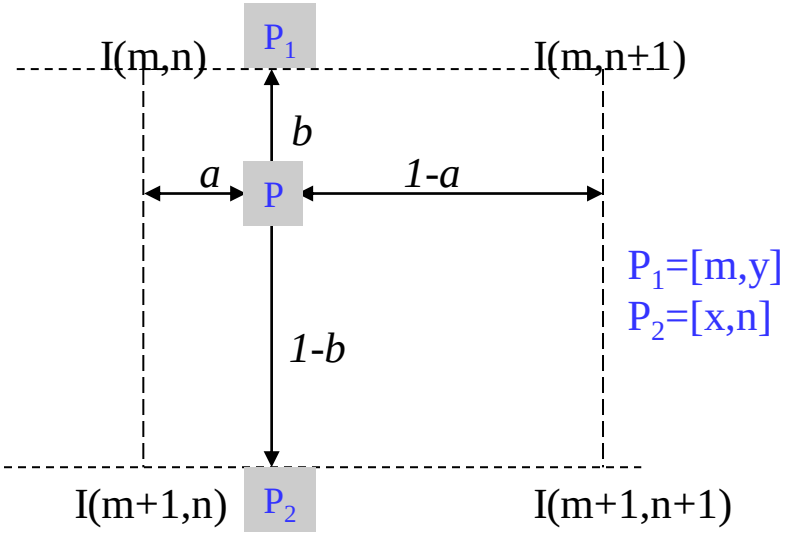


Interpolation method

1. Interpolate in 1 direction, say m

$I[P_1] = (1-a)I[m+1,n]+aI[m+1,n+1]$

$I[P_2] = (1-a)I[m,n]+aI[m,n+1]$

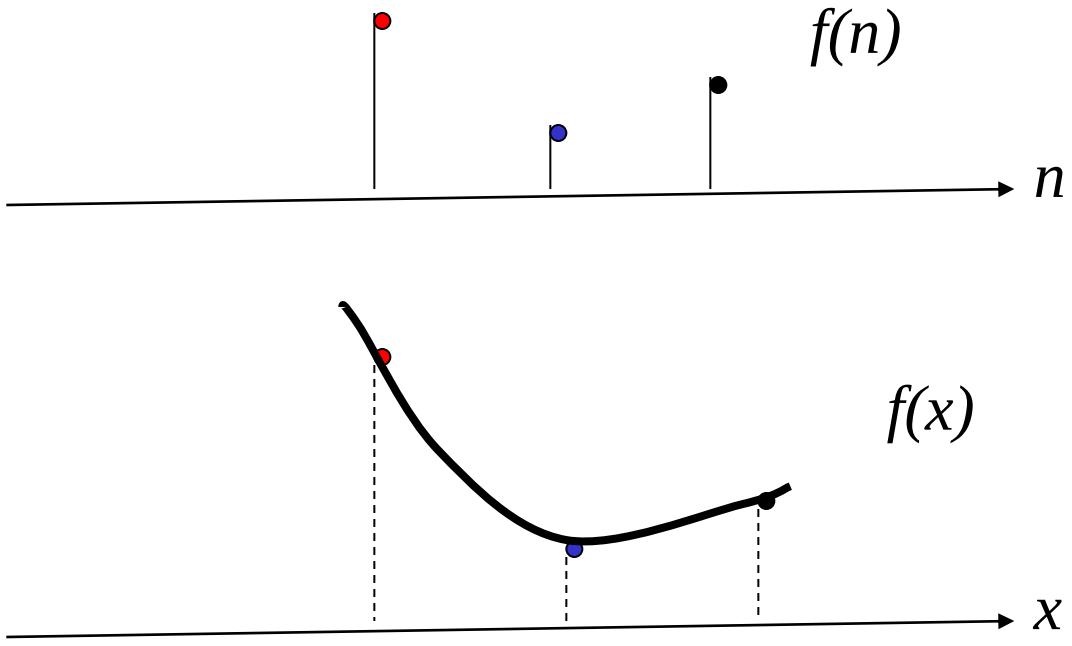


2. Interpolate the results in *n* direction

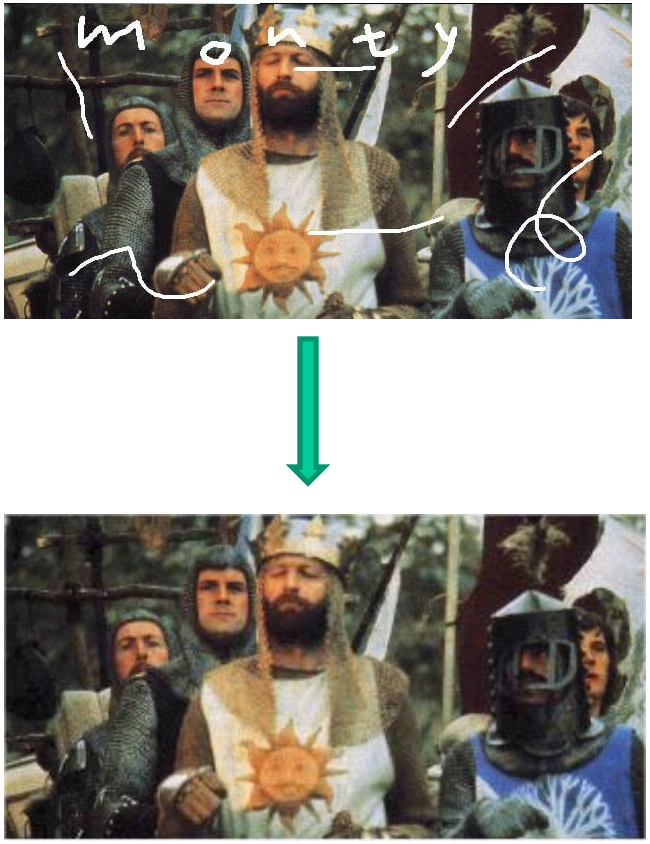
$I[P] = (1-b)I[P_1]+bI[P_2]$

Finally, $I[P]= (1-a)(1-b)I(m,n)+(1-a)bI(m+1,n) +a(1-b)I(m,n+1)+abI(m+1,n+1)$

Interpolation by *cubic spline* fitting



Other applications

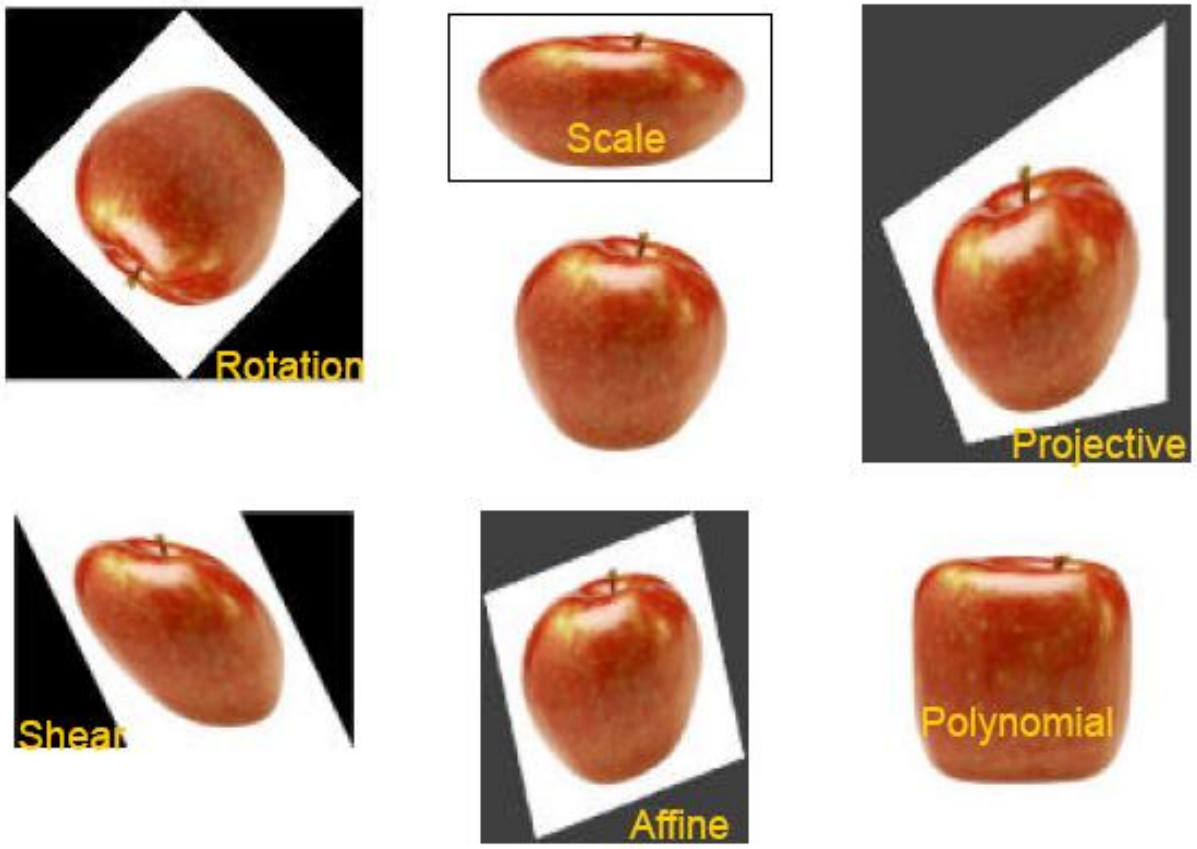


- Image inpainting
  - fill in missing or damaged pixels.

2. Geometric Transformation

- transforming the spatial variable

Geometric transformations



Regardless of the ‘distortions’ we see all the content as apples !

Geometric transformations

- Used mostly in image **correction**

- Pixel coordinate transformation

$$[m,n] \rightarrow [i,j]$$

$$[i,j] = T[m,n]$$

Where T is some transformation

Transformation categories

	Rotate	Translate	Scale	Skew/shear
Rigid	√	√	-	-
Similarity	√	√	√	-
Affine	√	√	√	√
Projective	√	√	√	√

Planar

- **Rigid** (or **Euclidean**) and **Similarity** **-** shapes are preserved
- **Affine** –Parallel lines remain parallel
- **Projective** – Most general type; angles between lines not preserved

Non-planar

- **Curved** – lines do not map to lines, shapes deform
  - Expressed as a displacement field or polynomials

Check

what type of *T* is applicable for the following?

- satellite image processing
- biometrics: face, fingerprint recognition
- license plate recognition
- sign language recognition

Basic transformations

T: Rotation

$$\begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} m \\ n \end{bmatrix}$$

T: Translation

$$\begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} m \\ n \end{bmatrix} + \begin{bmatrix} m_0 \\ n_0 \end{bmatrix}$$

T: Skew

$$\begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} m \\ n \end{bmatrix} + \begin{bmatrix} n\tan\theta \\ 0 \end{bmatrix}$$

T: Scaling

$$\begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} \alpha & \beta \end{bmatrix} \begin{bmatrix} m \\ n \end{bmatrix}$$

Combined transformations

- Rotation, translation, scaling, skewing

$$i = a_0 + a_1 m + a_2 n$$
$$j = b_0 + b_1 m + b_2 n$$

$$\Rightarrow \begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} a_1 & a_2 \\ b_1 & b_2 \end{bmatrix} \begin{bmatrix} m \\ n \end{bmatrix} + \begin{bmatrix} a_0 \\ b_0 \end{bmatrix}$$

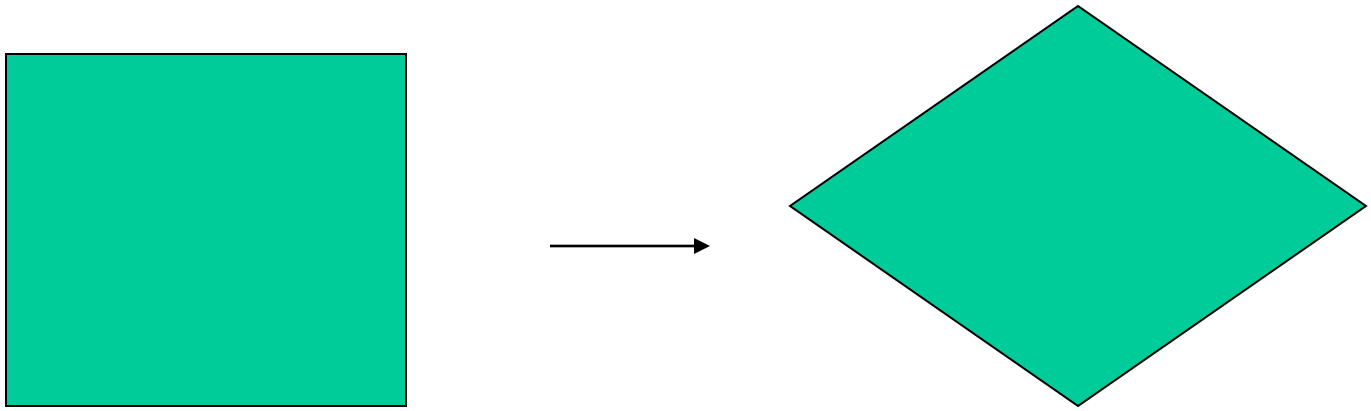
- *(i,j)* not guaranteed to be an integer

Scale

$a_1 = 1/2$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a_1 & 0 \\ 0 & 1/a_1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Affine Transform



square

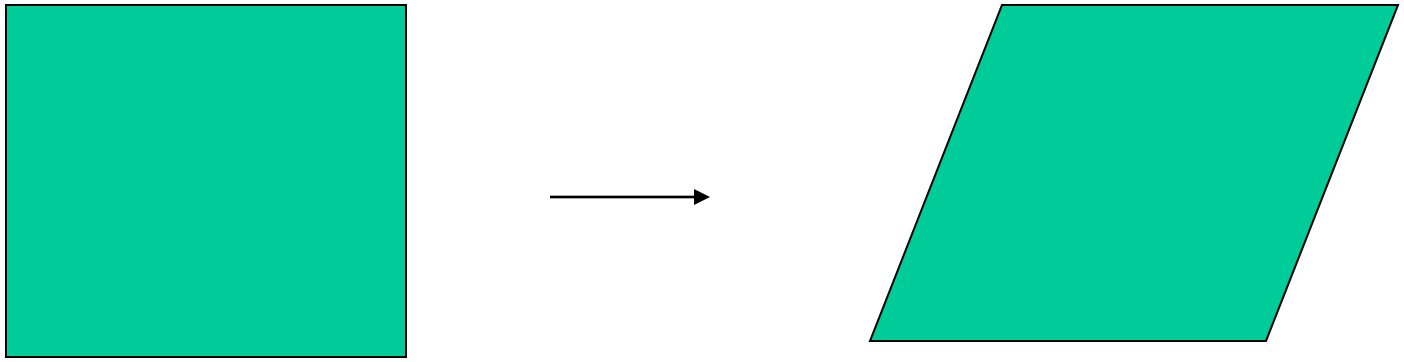
parallelogram

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a_1 & a_2 \\ b_1 & b_2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a_x \\ b_y \end{bmatrix}$$

Affine Transform

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} .5 & 1 \\ .5 & -2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Shear



square

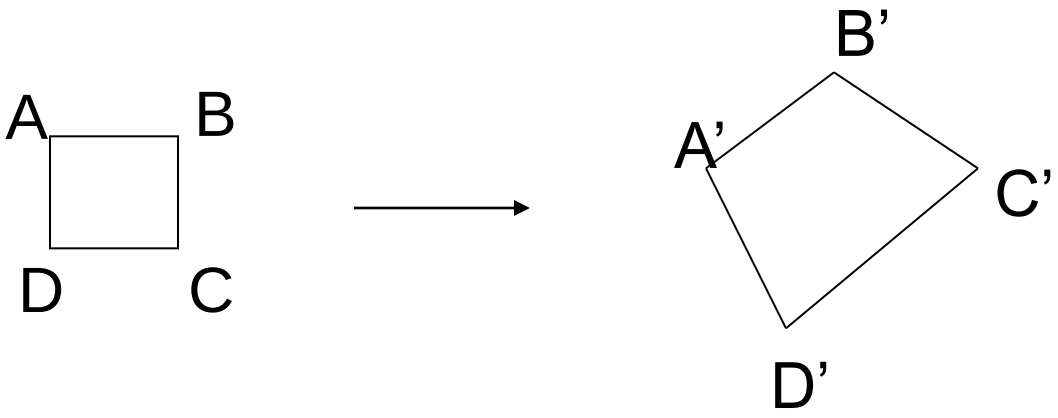
parallelogram

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ s & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a_x \\ b_y \end{bmatrix}$$

Shear M.Curie

What is the required affine matrix?

Projective (perspective) Transform

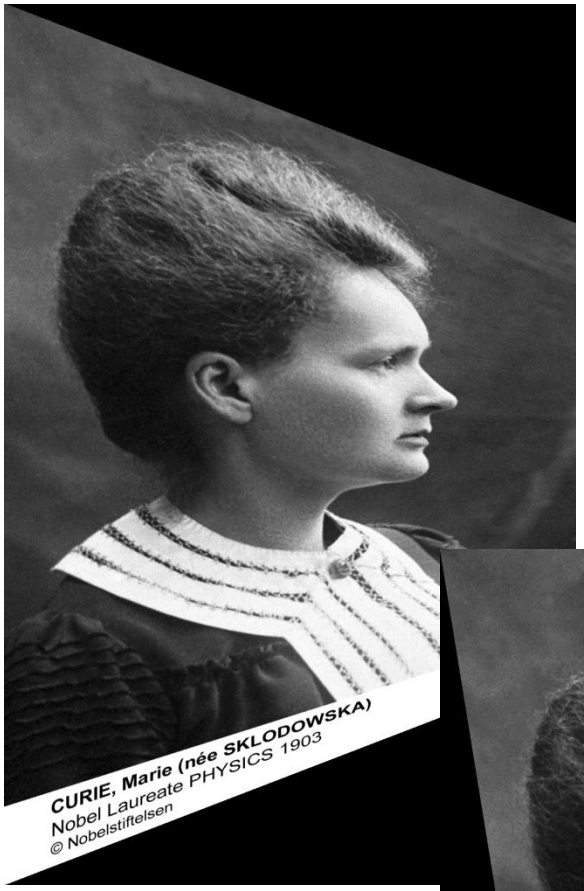


square quadrilateral

$$x' = \frac{a_1 x + a_2 y + a_3}{a_7 x + a_8 y + 1}$$

$$y' = \frac{a_4 x + a_5 y + a_6}{a_7 x + a_8 y + 1}$$

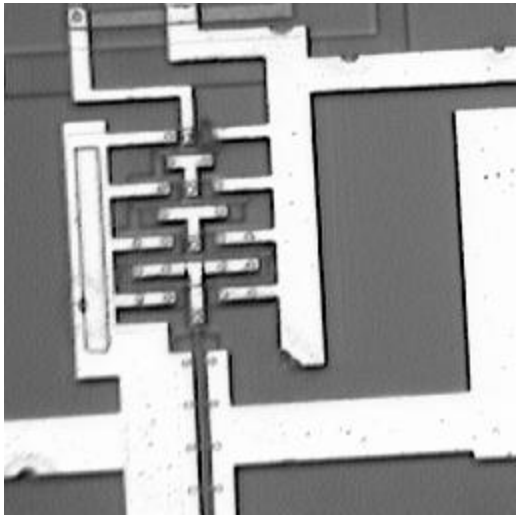
Perspective distortion

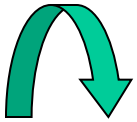


What is direction in these cases?

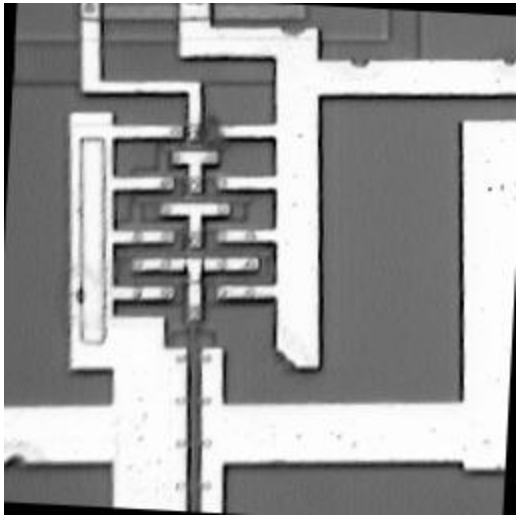
Issues in implementation

Example: Rotation

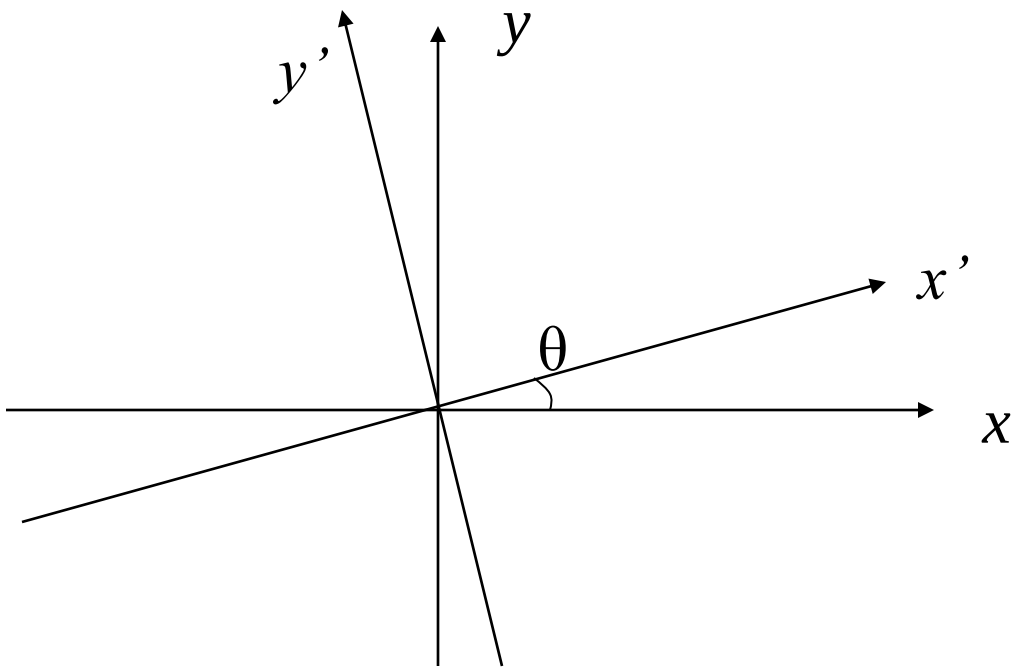




$\theta=3^{o}$



Rotation

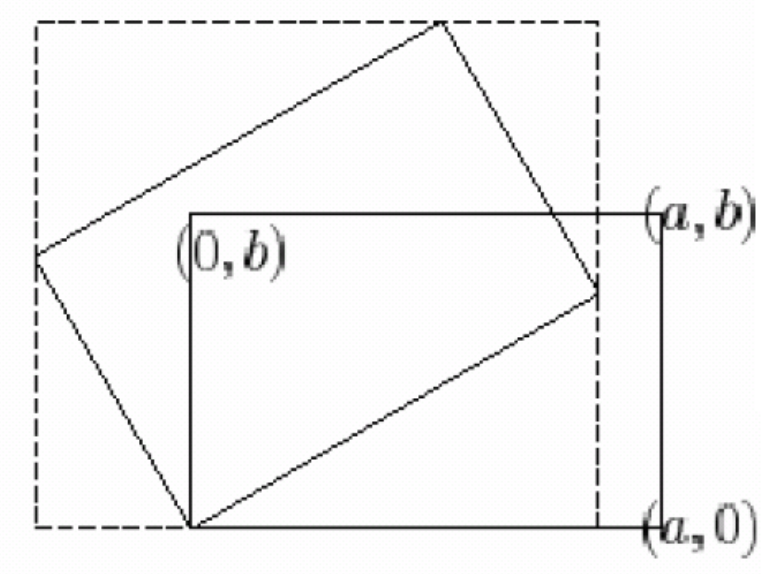


$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

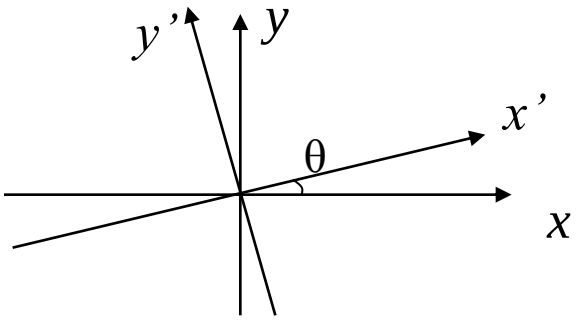
- Consider a rectangle of size *b* x *a*
- After rotation by θ, to capture the entire rectangle we need a larger size image (dashed lines).

Problem 1: some pixels lie outside the spatial range of the original rectangle.

Problem 2: some pixels end up with non-integer coordinates



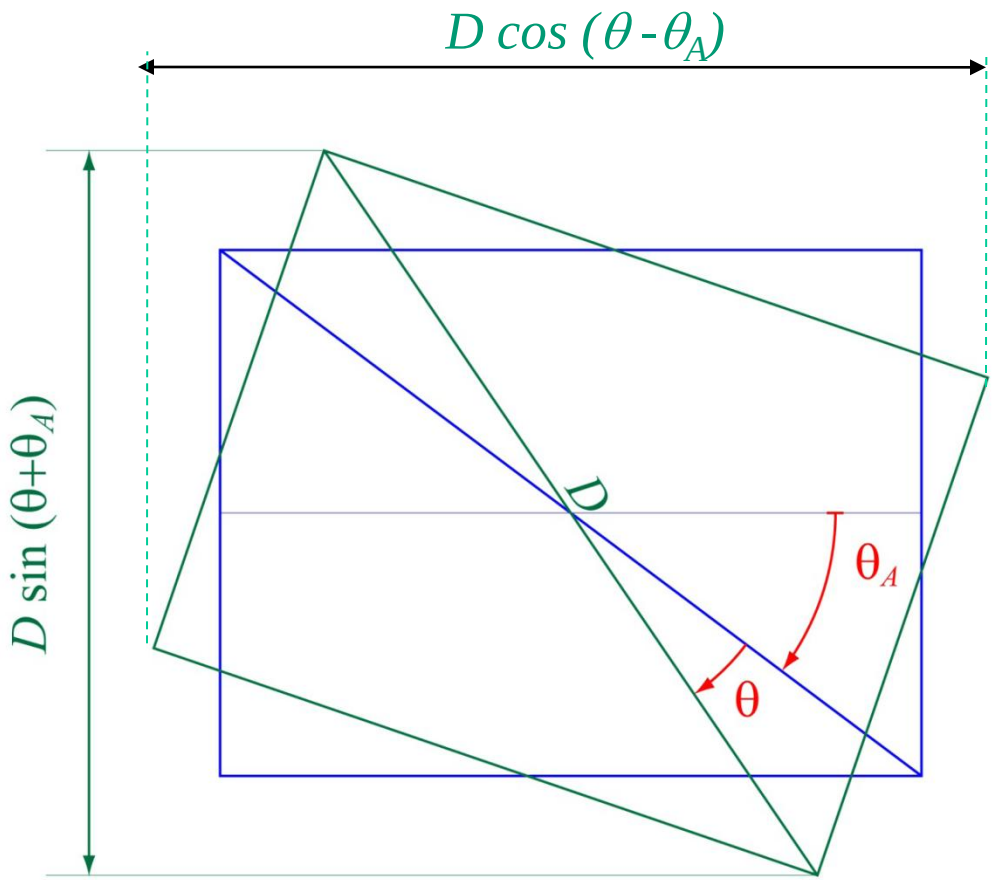
$$0 \le x'\cos\theta + y'\sin\theta \le a$$
$$0 \le -x'\sin\theta + y'\cos\theta \le b$$



Example

- b x a rectangle (in an image to be rotated by θ)
- Diagonal length $d = \sqrt{(a^2 + b^2)}$
- New image size: $b_{new} = ?$ $a_{new} = ?$
- Obviously this depends on θ!

Finding the size of the rotated image



Mapping the locations:
old image → new image ➔ Need to do interpolation

Computing the transformed image

$I[m,n] \rightarrow X[i,j]$;

$[i, j] = R \times [m, n] + t$: affine

1. Create a dummy matrix X[m, n] filled with zeros
2. For every [i,j] find the corresponding location [m,n] by applying the inverse transformation
   - $R^{-1}([i, j] - t) = [m, n]$
3. Find the pixel value at these source locations
   - For non-integer valued locations, use interpolation

Registration – an inverse problem

- A major problem involving geometric transformation is in spatial alignment of two images
- Given two images, $\mathbf{I_f[r]}$ and $\mathbf{I_m[r']}$ express their relation as $I_f[r] = I_m[T(r')]$
- Problem is to estimate T
  - make assumptions on nature of T to solve for it
  - generally done iteratively

So far..

- Given one image, we saw how to derive another image
  - by point or neighbourhood processing
  - by transforming the spatial variables

- Next, we will see how to derive an image by combining several images
  - Combination is via *arithmetic operations*

Image Arithmetic

- done as a tutorial
- here we see only applications

Image arithmetic

- Output image is derived from at least two input images using arithmetic operations

- Uses point operations

Common operations:

- addition
- subtraction
- multiplication
- division

http://www.aishack.in/2010/

Division operation in remote sensing

- Normalized Difference Vegetation Index (NDVI) is used to measure biomass

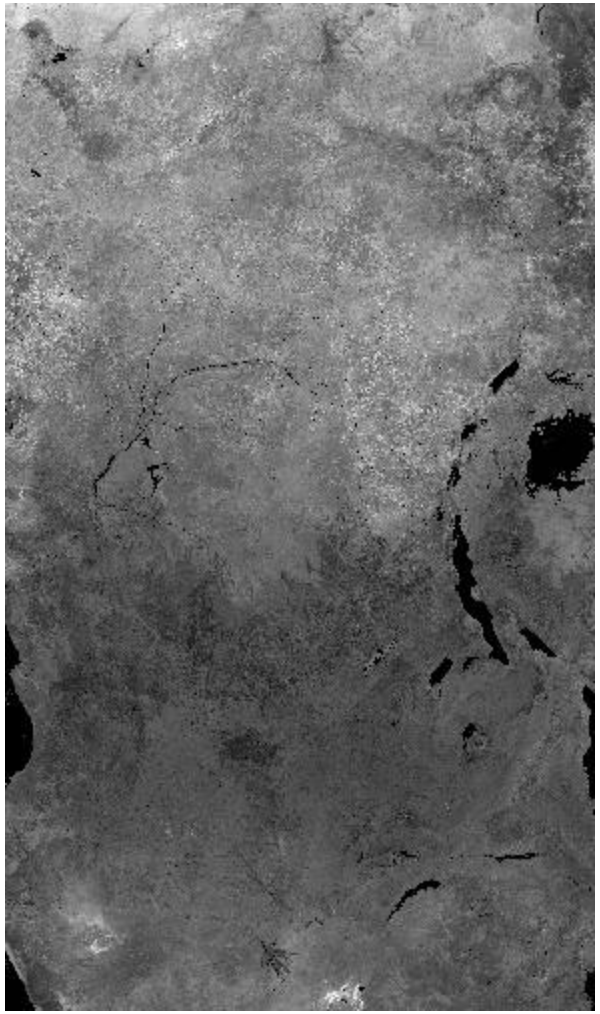
$$NDVI = \frac{NIR - R}{NIR + R}$$

NIR: Near Infrared band, R: Red; $-1 \leq NDVI \geq 1$

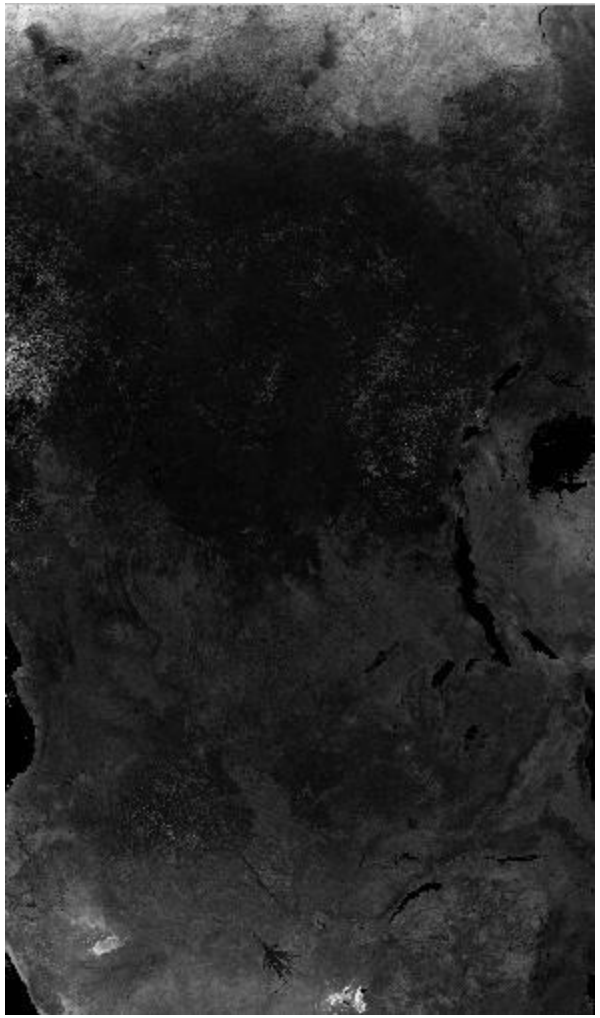
- Chlorophyll absorbs red light, therefore vegetation appears relatively darker in red band

- Useful to estimate crop production using satellite images

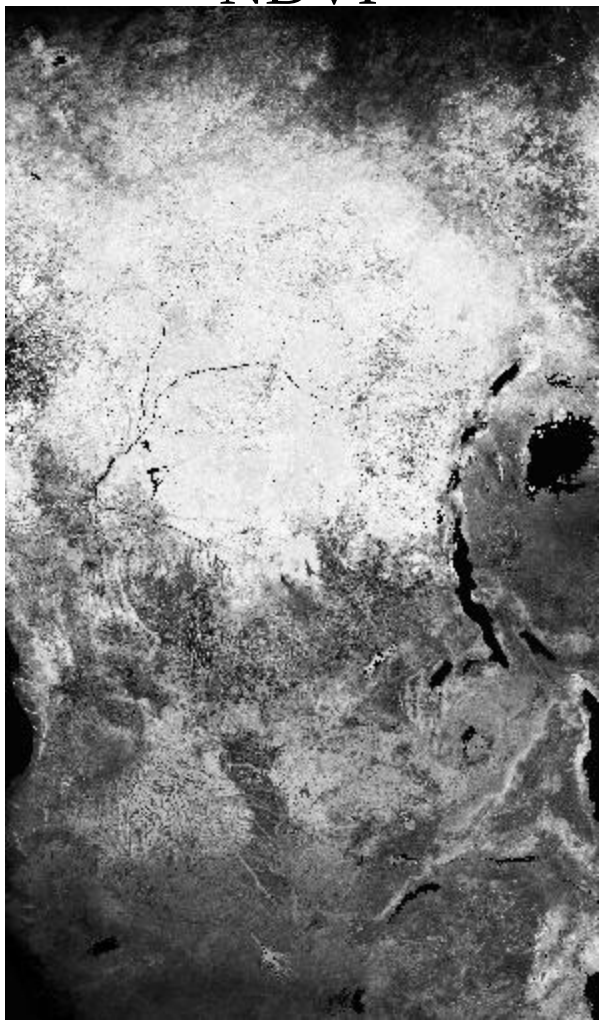
NIR



R



NDVI



Bright pixels represent vegetation

Source- Geo/SAT2, SUNY

Extracting objects of interest

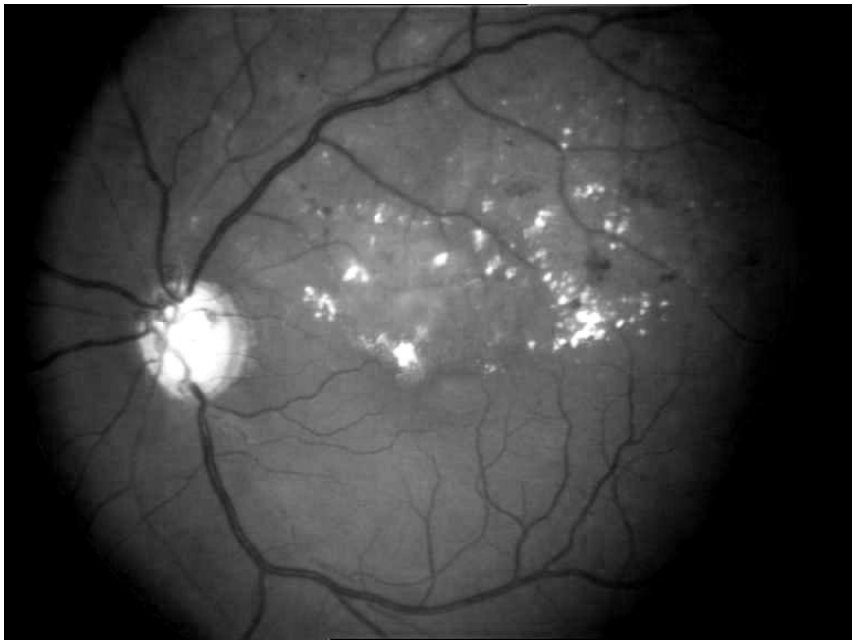
Problem: Given an image need to remove background

- Additive model: $I = foregnd + backgnd$

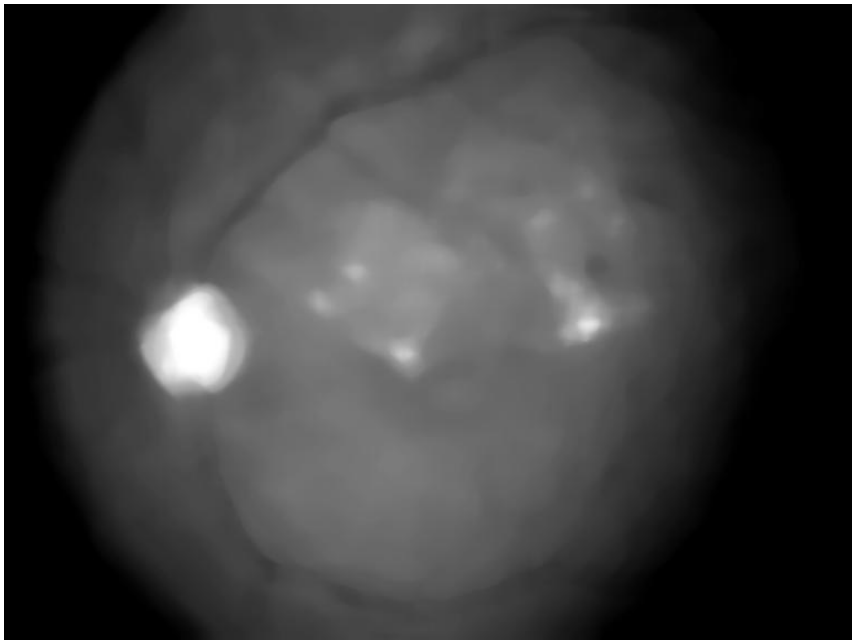
OR

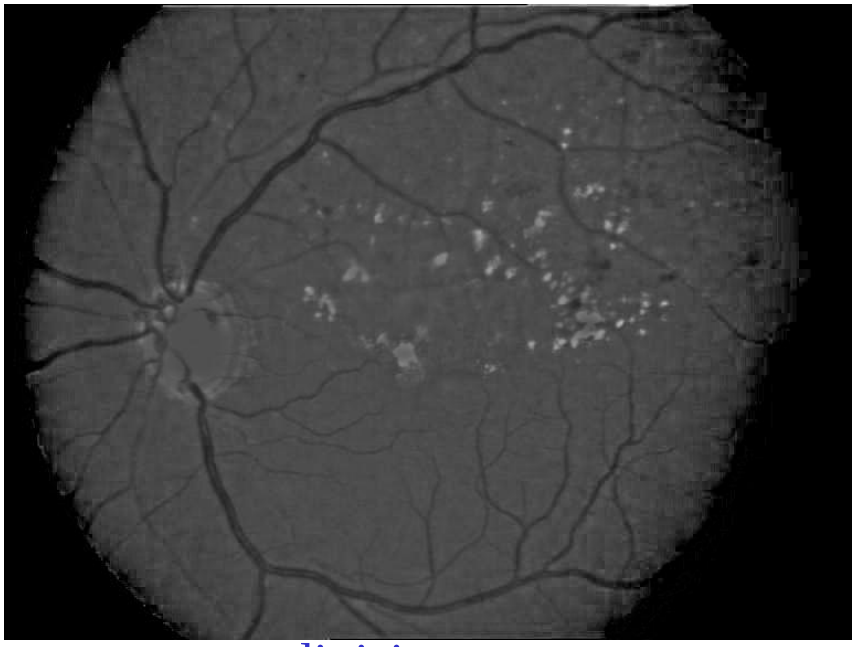
- Multiplicative model : $I = foregnd \; . \; backgnd$

input

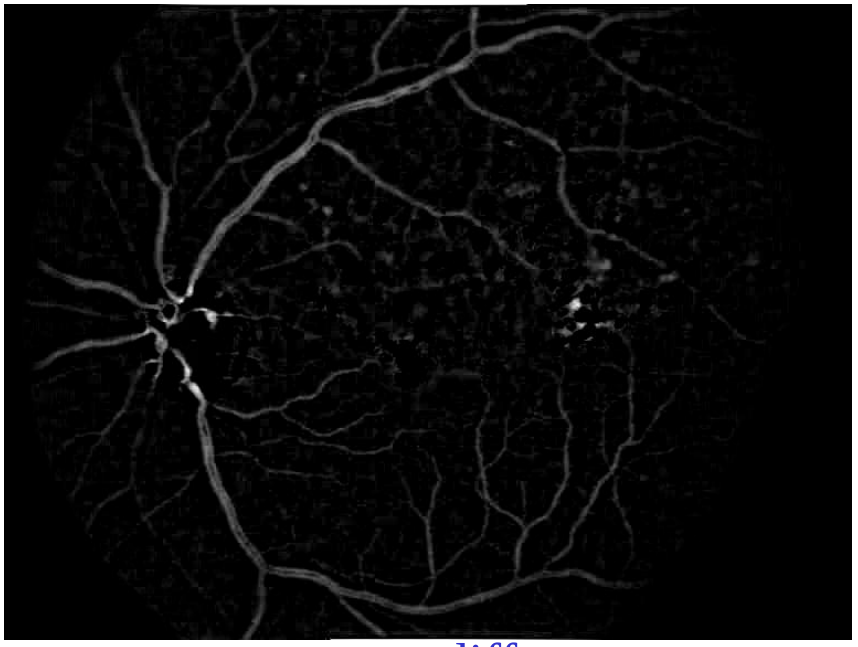


Estimated background





division



difference

Operational details

- *Background estimation*: median filter with large mask
- *Division*: Input / (Backgnd + 10)
  - mean shift by 10 needed to avoid divide by 0
  - Range is stretched to [0, 255] and quantized to 8 bit integer
- *Difference*: Input - background
  - Negative values have been clipped and range is stretched to [0,255]