# Introducing Assignment 1: Image Processing

COS 426: Computer Graphics (Fall 2022)

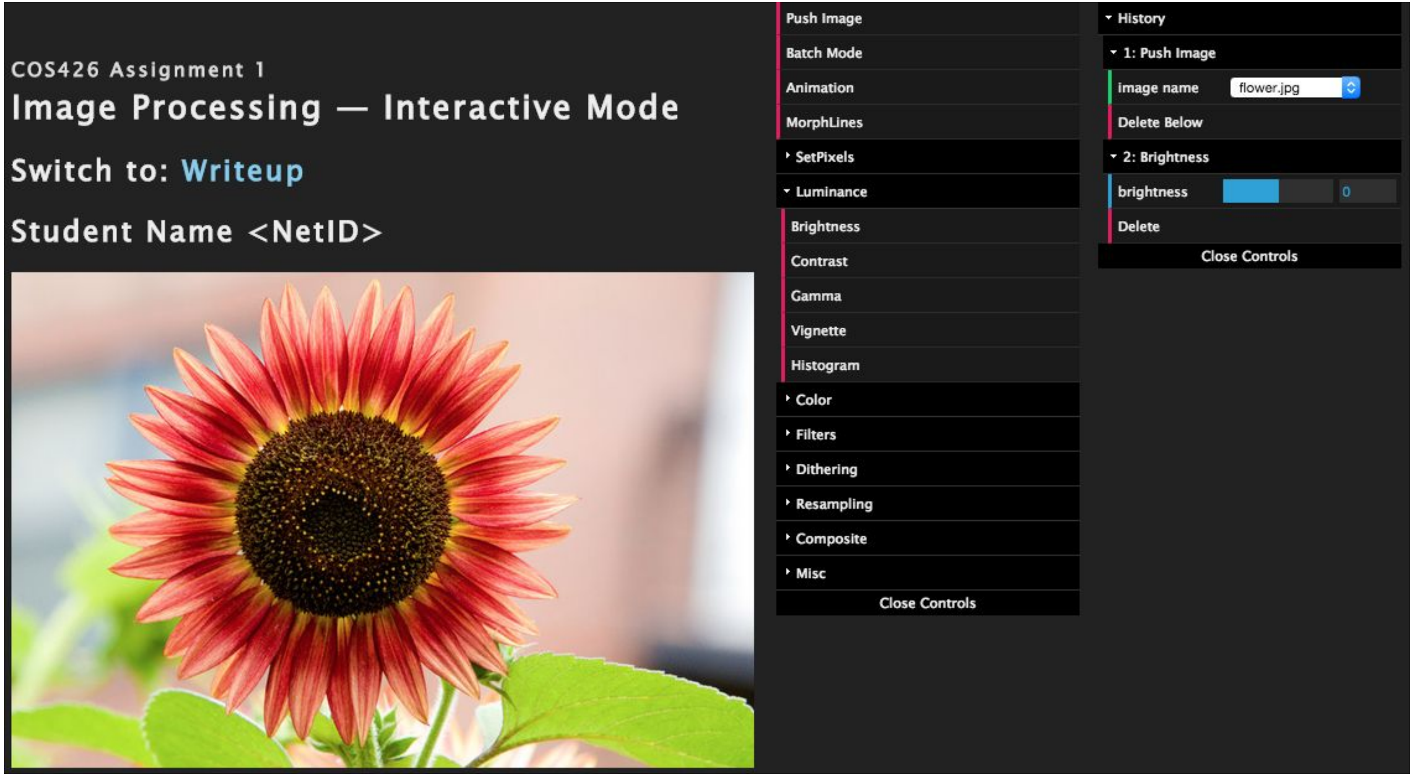
Guðni Nathan Gunnarsson, Yuanqiao Lin, Yuting Yang

# Setup

Same layout as A0:

- Run "`python3 -m http.server`" (or similar) inside the assignment directory
- Open "`http://localhost:8000`" in web browser

# GUI



COS426 Assignment 1
Image Processing — Interactive Mode
Switch to: Writeup
Student Name <NetID>
Push Image
Batch Mode
Animation
MorphLines
SetPixels
Luminance
Brightness
Contrast
Gamma
Vignette
Histogram
Color
Filters
Dithering
Resampling
Composite
Misc
Close Controls
History
1: Push Image
image name
flower.jpg
Delete Below
2: Brightness
brightness
0
Delete
Close Controls

# GUI

- Useful functions
  - Push Image
  - Animation: generate gif animation using (min, step, max)
  - MorphLines: specify line correspondences for morphing
  - BatchMode: fix current parameter settings

# GUI

- Features to implement
  - SetPixels: set pixels to certain colors (This was A0)
  - Luminance: change pixel luminance
  - Color: remap pixel colors
  - Filter: convolution/box filter
  - Dithering: reduce visual artifacts due to quantization ≈ cheat our eyes
  - Resampling: interpolate pixel colors
  - Composite: blending two images
  - Misc

# Features

Luminance
- Brightness
- Contrast
- Gamma
- Vignette
- Histogram equalization

Color
- Grayscale
- Saturation
- White balance
- Histogram matching

Filter
- Gaussian
- Sharpen
- Edge detect
- Median
- Bilateral filter

Dithering
- Quantization
- Random dithering
- Floyd-Steinberg error diffusion
- Ordered dithering

Resampling
- Bilinear sampling
- Gaussian sampling
- Translate
- Scale
- Rotate
- Swirl

Composite
- Composite
- Morph

Next week's precept will focus specifically on this topic

# A few reminders…

- Don’t try to exactly replicate example images.
- Choose parameters in your code which give you best looking results.
- Have fun!

# Changing Contrast

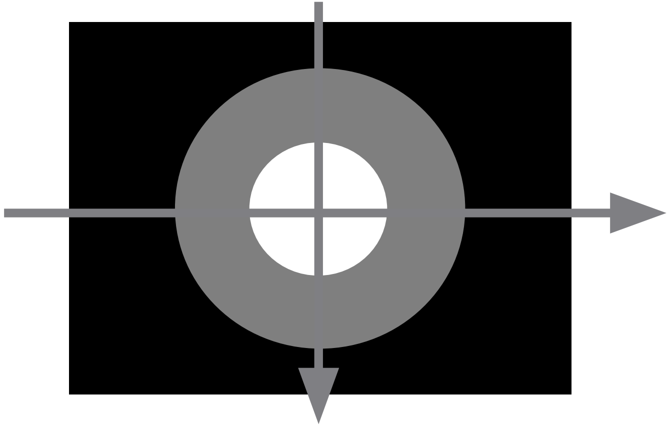
- GIMP formula
  - value = (value - 0.5) * (tan ((contrast + 1) * PI/4) ) + 0.5;
  - "Difference above mid-value times contrast multiplier, plus mid-value”
  - When contrast=1, tan(PI/2) is infinite, think about limit and what is reasonable
  - Clamp pixel to [0, 1] after computing the value.
  - Apply to each channel separately.

# Gamma correction

- R = R^gamma, G = G^gamma, B = B^gamma
- R,G,B are typically in [0, 1] (default in the code base)
- Second arg of gammaFilter(image, logOfGamma) is log(gamma)
  - So use gamma = Math.exp(logOfGamma)
- Exponentiation in JS is “Math.pow(base, exponent)” or (ES7 / ES2017+) “base**pow”
  - Your browser might not support ES7

# Vignette

- Pixels within inner radius remain unchanged
- Pixels outside outer radius are black
- Pixels between innerR and outerR should be multiplied with a value in [0, 1]:
  - R = sqrt(x^2 + y^2) / halfdiag
  - Multiplier = 1 - (R - innerR) / (outerR - innerR)
- Similar to soft brush



Multiplier map

# Histogram Equalization

Transform an image so that it has flat histogram of luminance values.

Before

After

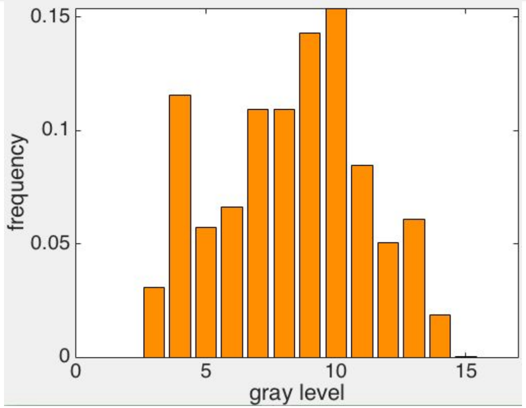
# Histogram Matching

Transform an image so that it has same histogram of luminance values as reference image.

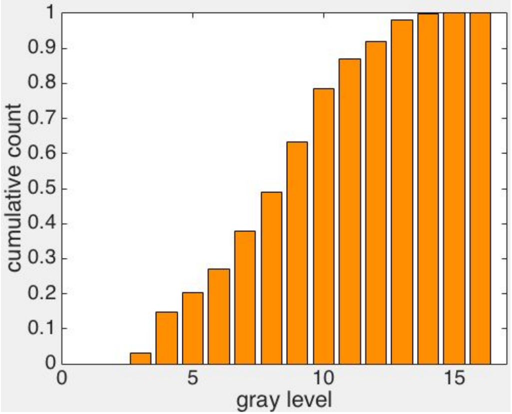
*reference image: town*

*reference image: flower*

# Histogram Equalization/Matching



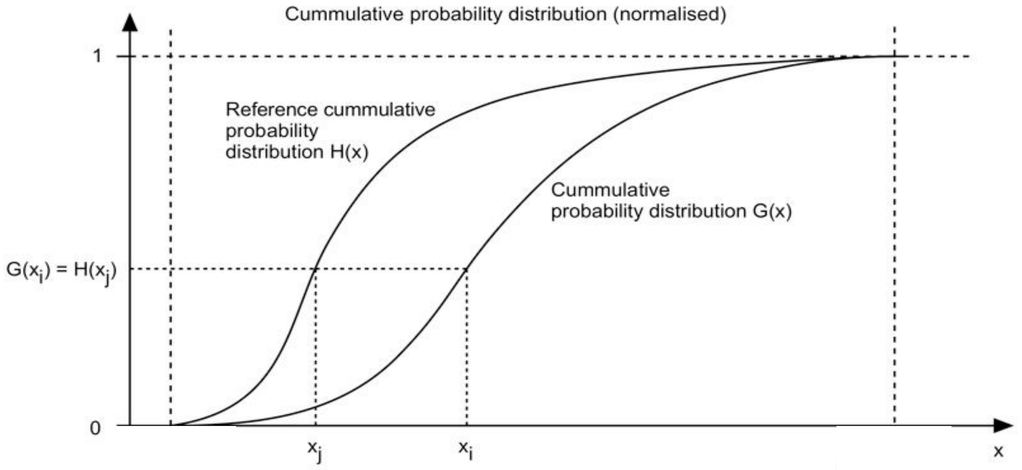

pdf




cdf

# Histogram Equalization/Matching

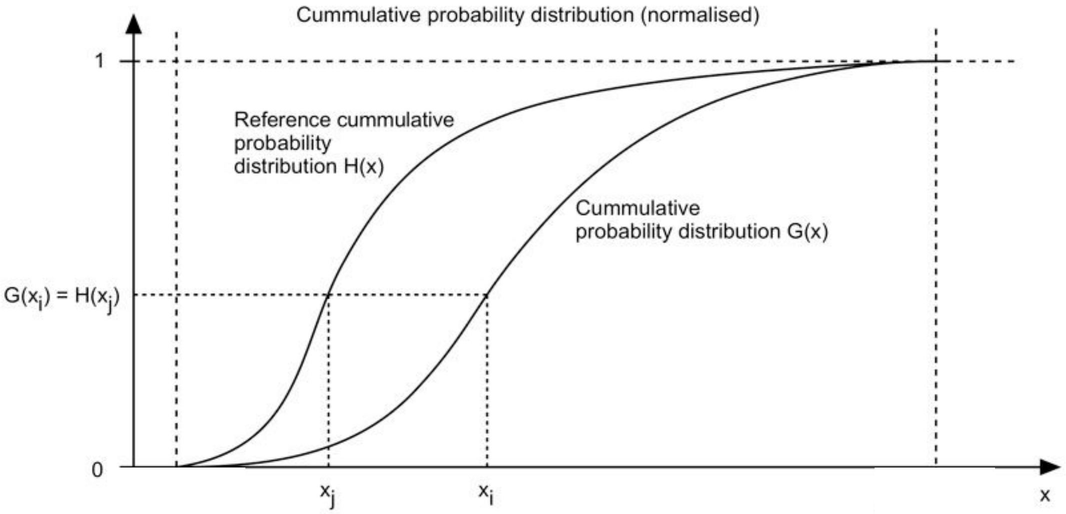
- Image: x
- Number of gray levels: L
- $pdf(i) = \frac{n_i}{n}$   $n_i$ = number of pixels of the i-th gray level
- $cdf(j) = \sum_{j=0}^{i} pdf(i)$
- Target cdf:
  - Equalization:
    - $cdf_{ref}(i) = \frac{i}{L-1}$
  - Matching:
    - cdf of the reference image




(source:http://paulbourke.net/miscellaneous/equalisation/)

# Histogram Equalization/Matching

- Target cdf:
  - Equalization:
    - $cdf_{ref}(i) = \frac{i}{L-1}$
  - Matching:
    - cdf of the reference image
- Implementation
  - Equalization
    - $x' = (cdf(x) * (L-1)) / (L-1)$
  - Matching
    - $x' = arg\min_{i} |cdf(x) - cdf_{ref}(i)|$
    - Convert back to gray level: $x' = \frac{x\prime}{L-1}$



# Saturation

- pixel = pixel + (pixel - gray(pixel)) * ratio
- Do clamp()

-1 -0.5 0 0.5 1.0

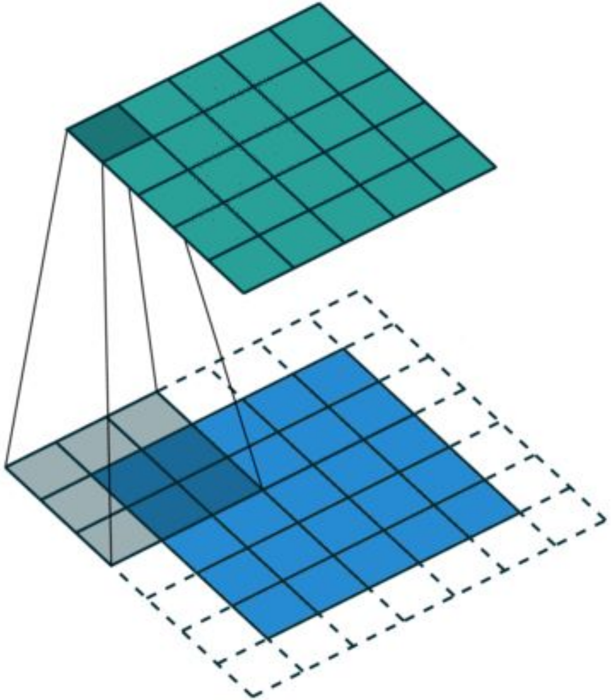
# White balance

```
whitebalance(image, rgb_w)
    [L_w, M_w, S_w] = rgb2lms(rgb_w)
    for each pixel x in image
        [L, M, S] = rgb2lms(image(x))
        L = L / L_w
        M = M / M_w
        S = S / S_w
        image_out(x) = lms2rgb(L, M, S)
```

- Hints:
  - Use rgbToXyz(), xyzToLms(), lmsToXyz(), xyzToRgb()
  - Do clamp()

# Convolution (Gaussian/Sharpen/Edge)

| | | |
|---|---|---|
| w1 | w2 | w3 |
| w4 | w5 | w6 |
| w7 | w8 | w9 |



# Convolution (Gaussian/Sharpen/Edge)

- Weights can be normalized depending on the application
- Variety of ways to handle edges
  - Mirror boundary
  - Zero padding
  - Use part of the kernel only

# Gaussian filter

- Create a new image to work on
- Weights should be normalized to sum to 1, otherwise average color changes

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

  - x = distance to the center of the kernel
- Linear separation optimization:
  - First apply a 1D Gaussian kernel vertically and then a 1D Gaussian kernel horizontally

# Edge

Kernel:

| | | |
|---|---|---|
| -1 | -1 | -1 |
| -1 | 8 | -1 |
| -1 | -1 | -1 |

Inside boundary

| | |
|---|---|
| 3 | -1 |
| -1 | -1 |

At boundary

- Weights sum to 0
- Optional to invert the edge map for visualization:
- pixel = 1 - pixel

# Sharpen

- Kernel:

| | | |
|---|---|---|
| -1 | -1 | -1 |
| -1 | 9 | -1 |
| -1 | -1 | -1 |

Inside boundary

| | | |
|---|---|---|
| | | |
| | 4 | -1 |
| | -1 | -1 |

At boundary

- Weights sum to 1

# Edge Filter vs Sharpen Filter

| | | |
|---|---|---|
| **-1** | **-1** | **-1** |
| **-1** | **8** | **-1** |
| **-1** | **-1** | **-1** |

Edge Filter

| | | |
|---|---|---|
| **-1** | **-1** | **-1** |
| **-1** | **9** | **-1** |
| **-1** | **-1** | **-1** |

Sharpen Filter

Convolution(Image, Sharpen Filter) = Convolution(Image, Edge Filter) + Image

# Median

- Use a window (similar to convolution)
- Choose the median within the window
- Sorting: sort by RGB separately / sort by luminance
- Optimization: use quick-select to find median
  - Gives median in linear time

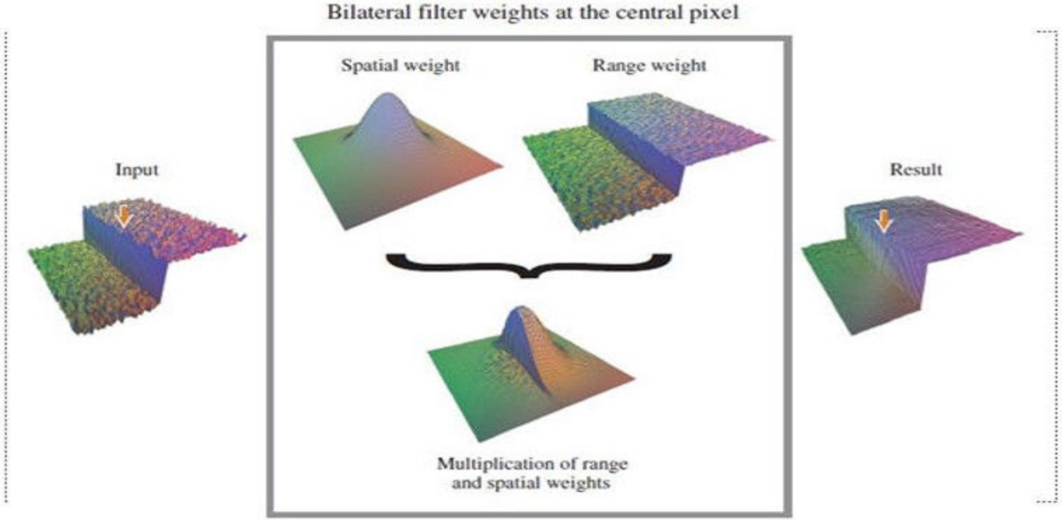

RGB Example

# Bilateral

- Combine Gaussian filtering in both spatial domain and color domain
- Weight formula of filter for pixel (i, j): Spatial distance component  Color distance component

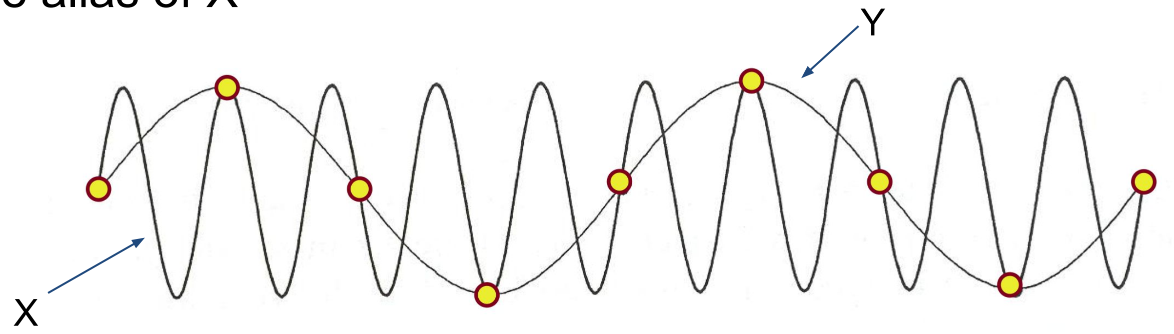
$$w(i,j,k,l) = e^{\left(-\frac{(i-k)^2+(j-l)^2}{2\sigma_d^2} - \frac{\|I(i,j)-I(k,l)\|^2}{2\sigma_r^2}\right)}$$

- Similar color -> large weights, Different color -> smaller weights

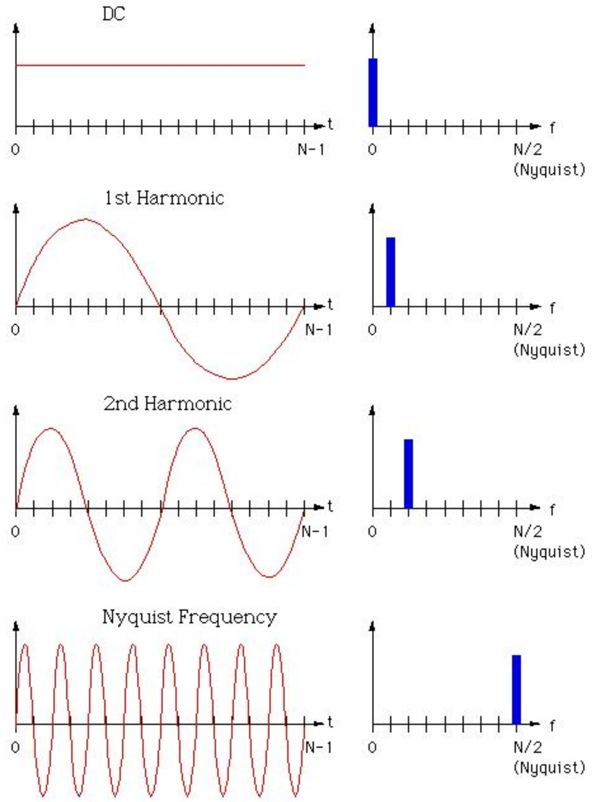


# Sampling & Frequencies

- Real-world is continuous, Sensors are discrete
- How many samples do we need to measure real world?
  - Too few samples = aliasing
  - Nyquist rate says that we need to sample at ≥ 2× the highest frequency for perfect reconstruction
- Aliasing is when signal X masquerades as signal Y
  - Y is the alias of X

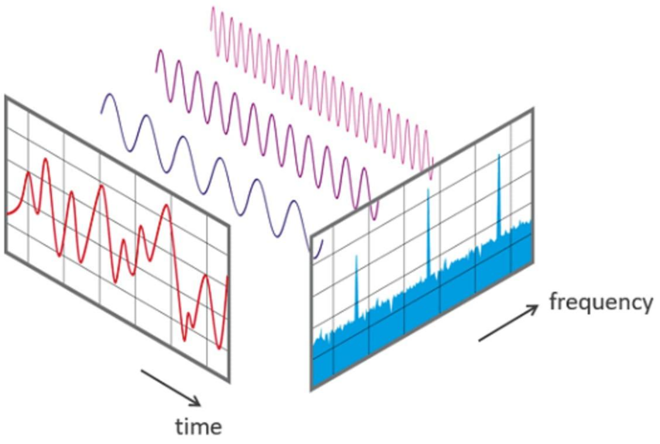


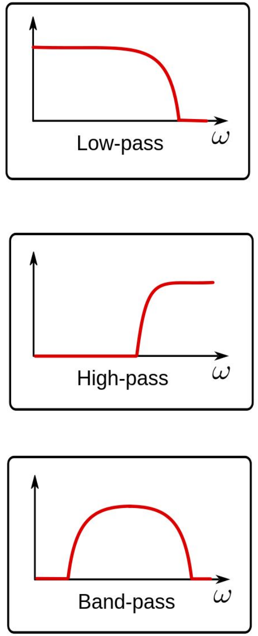
# Fourier Transform



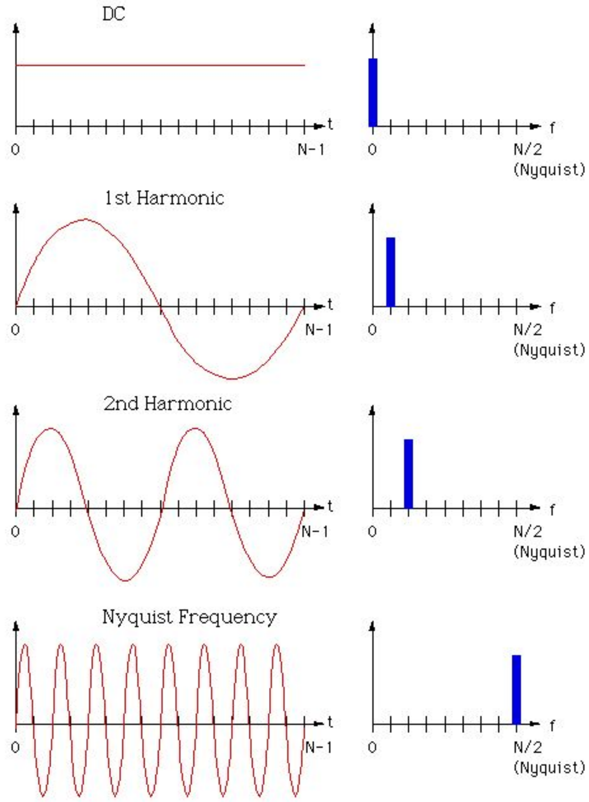

Maps signal from time domain to frequency domain

Use low-pass filter to remove high frequencies and prevent aliasing



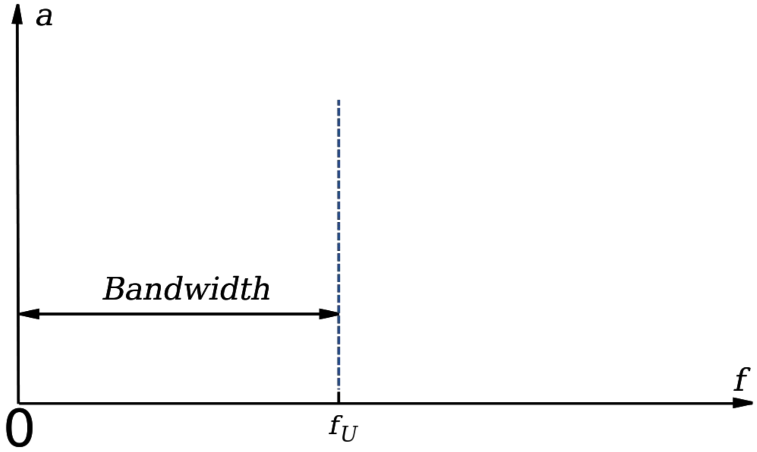
# Fourier Transform




Maps signal from time domain to frequency domain

Use low-pass filter to remove high frequencies and prevent aliasing

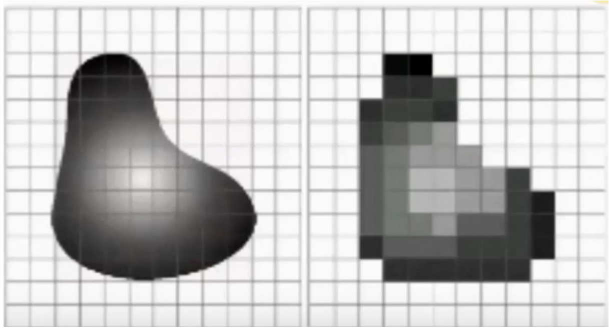
Bandlimiting



# 1D to 2D

- 2D signals follow the same analysis as 1D signals

Real world 2D image is sampled by sensor



Aliasing for 2D signals

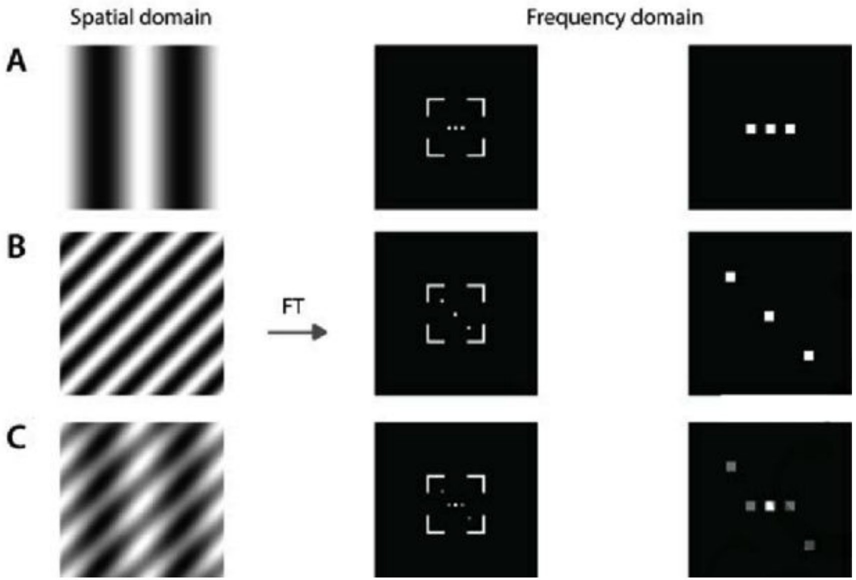
(Barely) adequate sampling

Inadequate sampling

# 1D to 2D

- 2D signals follow the same analysis as 1D signals

Fourier Analysis for 2D signals




If image resolution is low
- E.g. image compression

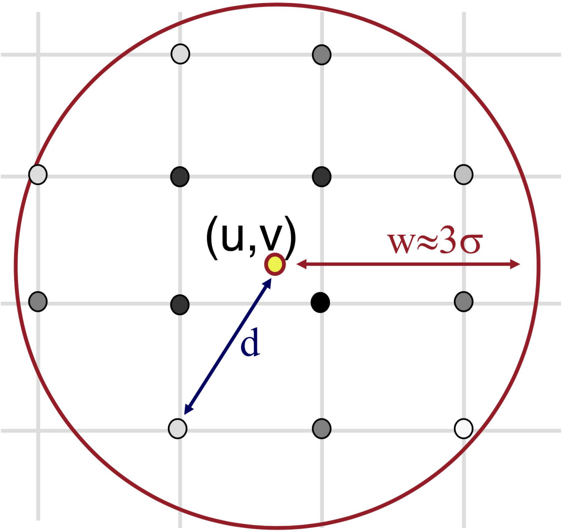
Then need to apply band-limiting filter to avoid aliasing
- E.g. Triangle, Gaussian

Note that these filters are “finite” filters, they act as approximations to a perfect low pass filter

# Resampling

- Gaussian interpolation
  - Weights:

$$G(d,\sigma) = e^{-d^2/(2\sigma^2)}$$



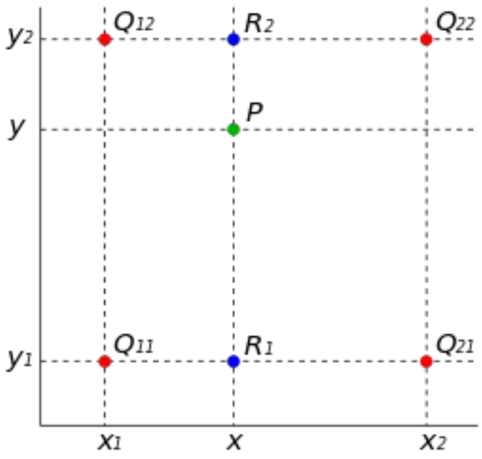

- Weights need to be normalized, so that sum up to 1
- Use windowSize = 3*sigma
  - Sigma can be 1
- Window can be square

# Resampling

- Bilinear interpolation

$$f(x,y) = \frac{1}{(x_2 - x_1)(y_2 - y_1)} \big(f(Q_{11})(x_2 - x)(y_2 - y) + f(Q_{21})(x - x_1)(y_2 - y) + f(Q_{12})(x_2 - x)(y - y_1) + f(Q_{22})(x - x_1)(y - y_1)\big)$$

(from wikipedia)



# Quantization

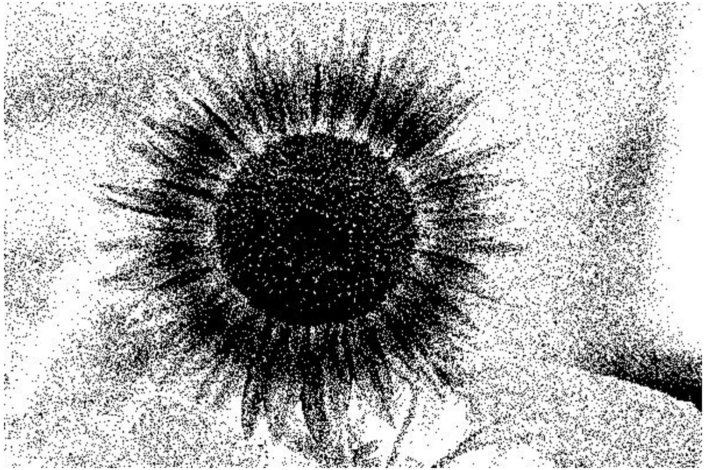
- Quantize a pixel within [0, 1] using n bits
  - round(p * (2^n-1)) / (2^n-1)

n=1 example

# Random dithering

- Before quantization:
  - p = p + (random() - 0.5)/(2^n-1)
  - n is number of bits per channel

Reduce banding with intentional noise



n=1 example

# Ordered dithering

**Pseudo code for n-bit case:**

```
i = x mod m
j = y mod m
err  = I(x, y) - floor_quantize(I(x, y)))
threshold = (D(i, j)+ 1) / (m^2 + 1)
if err > threshold
    P(x, y) = ceil_quantize(I(x, y)))
else
    P(x, y) = floor_quantize(I(x, y)))
```

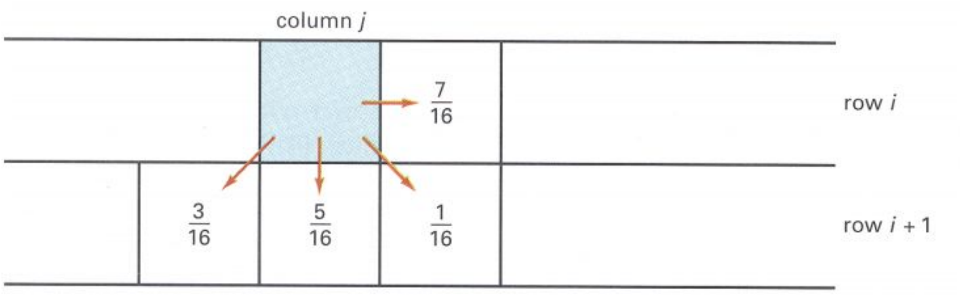
- `floor_quantize(p)`
  `= floor(p * (2^n-1)) / (2^n-1)`
- `ceil_quantize(p)`
  `= ceil(p * (2^n-1)) / (2^n-1)`

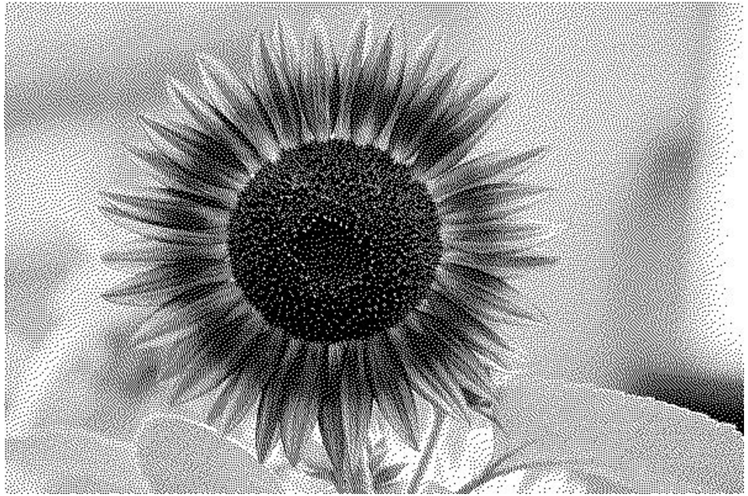
m = 4, D= $\begin{bmatrix} 15 & 7 & 13 & 5 \\ 3 & 11 & 1 & 9 \\ 12 & 4 & 14 & 6 \\ 0 & 8 & 2 & 10 \end{bmatrix}$

n=1 example

# Floyd-Steinberg error diffusion

- Loop over pixels line by line
  - Quantize pixel
  - Compute quantization error (the difference of the original pixel and the quantized pixel)
  - Spread quantization error over four unseen neighboring pixels with weights (see left figure below)
- Results look more natural

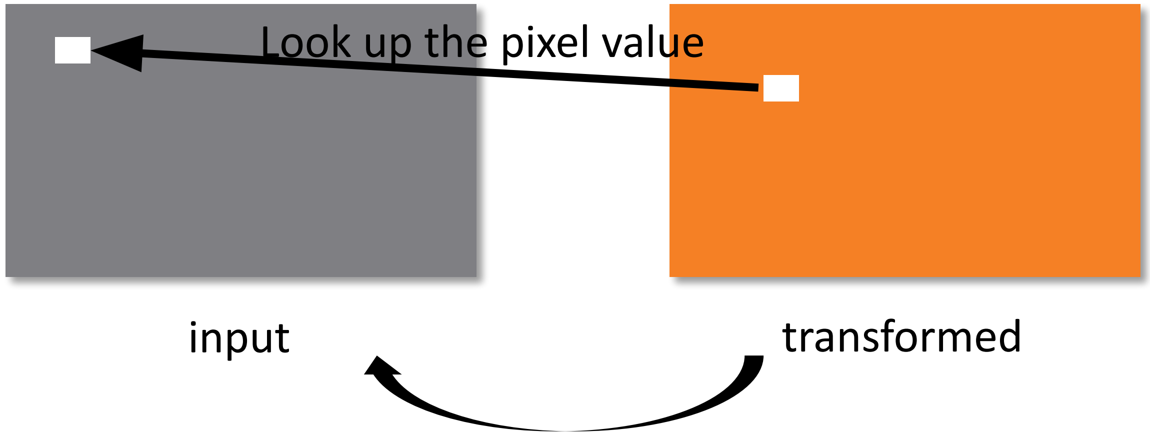


# Q&A

# Transformation (translate/scale/rotate/swirl)

- Inverse mapping




Inverse mapping guarantees that every pixel in the transformed image is filled!

# Transformation (translate/scale/rotate/swirl)

- To fill in a pixel in the target image, apply the inverse transform to the pixel location and look it up in the input image (with resampling technique) for pixel value.
- i.e. For translation of $x' = x + tx$, $y' = y + ty$:

  $I'(x', y') = I(x' - tx, y' - ty)$

- i.e. For scale of $x' = x * sx$, $y' = y * sy$:

  $I'(x', y') = I(x' / sx, y' / sy)$

# Composite

- output = alpha * foreground + (1 - alpha) * background
- alpha is the alpha channel foreground

backgroundImg

foregroundImg

foregroundImg(alpha channel)

Result

# Morph

- Basic concepts
  – transform the background image to the foreground image
  – alpha = 0: show background
  – alpha = 1: show foreground
  – alpha is the blending factor / timestamp
- General approach
  – specify correspondences (morphLines.html)
  – create an intermediate image with interpolated correspondences (alpha)
  – warp the background image to the intermediate image
  – warp the foreground image to the intermediate image
  – blend using alpha

# Morph

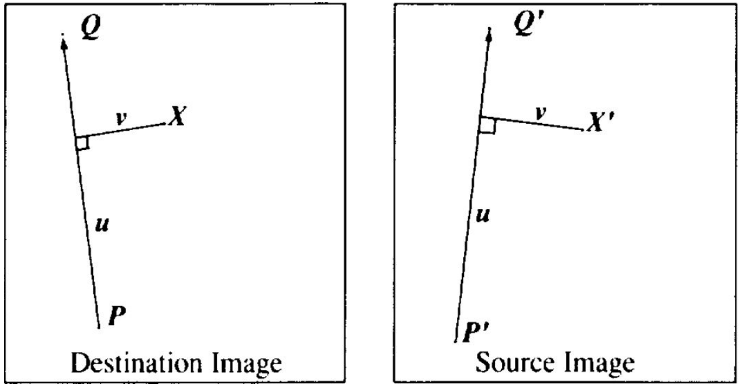
```
GenerateAnimation(Image_0, L_0[…], Image_1, L_1[…])
begin
    foreach intermediate frame time t do
        for i = 0 to number of line pairs do
            L[i] = line t-th of the way from L_0[i] to L_1[i]
        end
        Warp_0 = WarpImage(Image_0, L_0, L)
        Warp_1 = WarpImage(Image_1, L_1, L)
        foreach pixel p in FinalImage do
            Result(p) = (1-t) Warp_0 + t Warp_1
        end
    end
end
```

# Warp Image

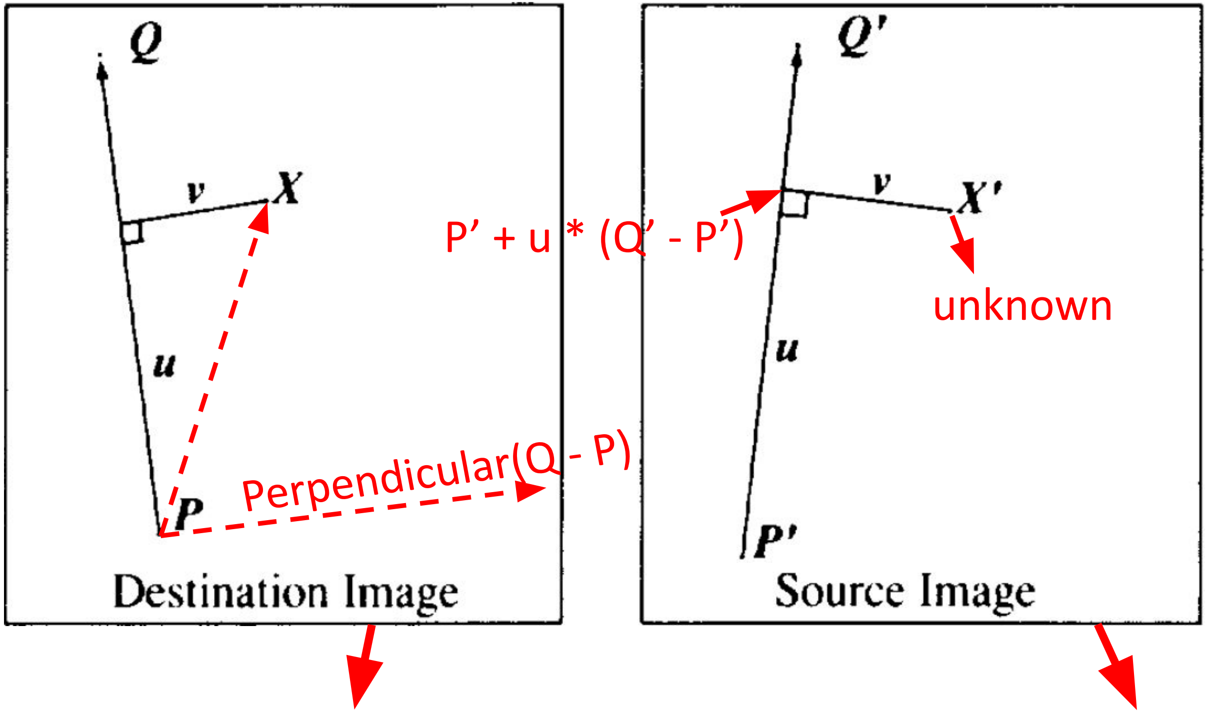
- $u = \frac{(X-P)\cdot(Q-P)}{||Q-P||^2}$
- $v = \frac{(X-P)\cdot Perpendicular(Q-P)}{||Q-P||}$ unit vector
- $X' = P' + u \cdot (Q' - P') + \frac{v \cdot Perpendicular(Q'-P')}{||Q'-P'||}$ unit vector
- $dist = shortest\ distance\ from\ X\ to\ PQ$
  - 0 <= u <= 1: dist = |v|
  - u < 0: dist = ||X – P||
  - u > 1: dist = ||X – Q||
- $weight = (\frac{length^p}{a+dist})^b$
  - we use p = 0.5, a = 0.01, b = 2

Contribution of line segment PQ to the warping of X's location

If Q - P = (x, y),
Perpendicular(Q – P) = (y, -x)



# Warp Image

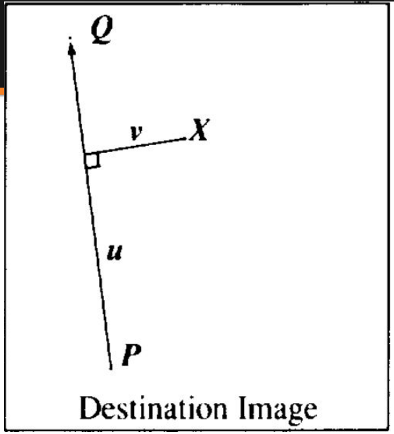


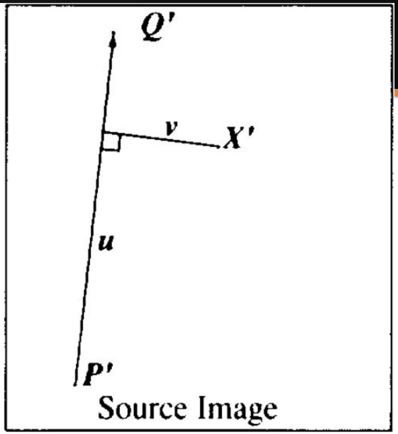

Warped background or foreground (currently black)

Pixel source (background or foreground)

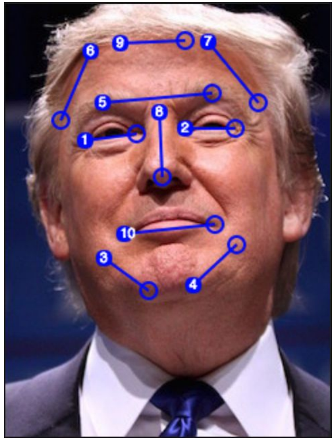
# Warp Image

For each pixel $X$ in the destination
    $DSUM = (0,0)$
    $weightsum = 0$
    For each line $P_i Q_i$
        calculate $u,v$ based on $P_i Q_i$
        calculate $X'_i$ based on $u,v$ and $P_i'Q_i'$
        calculate displacement $D_i = X_i' - X_i$ for this line
        $dist$ = shortest distance from $X$ to $P_i Q_i$
        $weight = (length^p / (a + dist))^b$
        $DSUM$ += $D_i * weight$
        $weightsum$ += $weight$
    $X' = X + DSUM / weightsum$
    destinationImage($X$) = sourceImage($X'$)

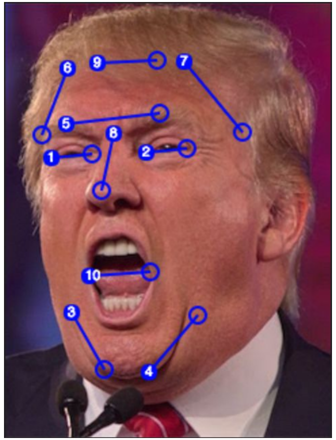


# Interpolate Morph Lines




Background Image

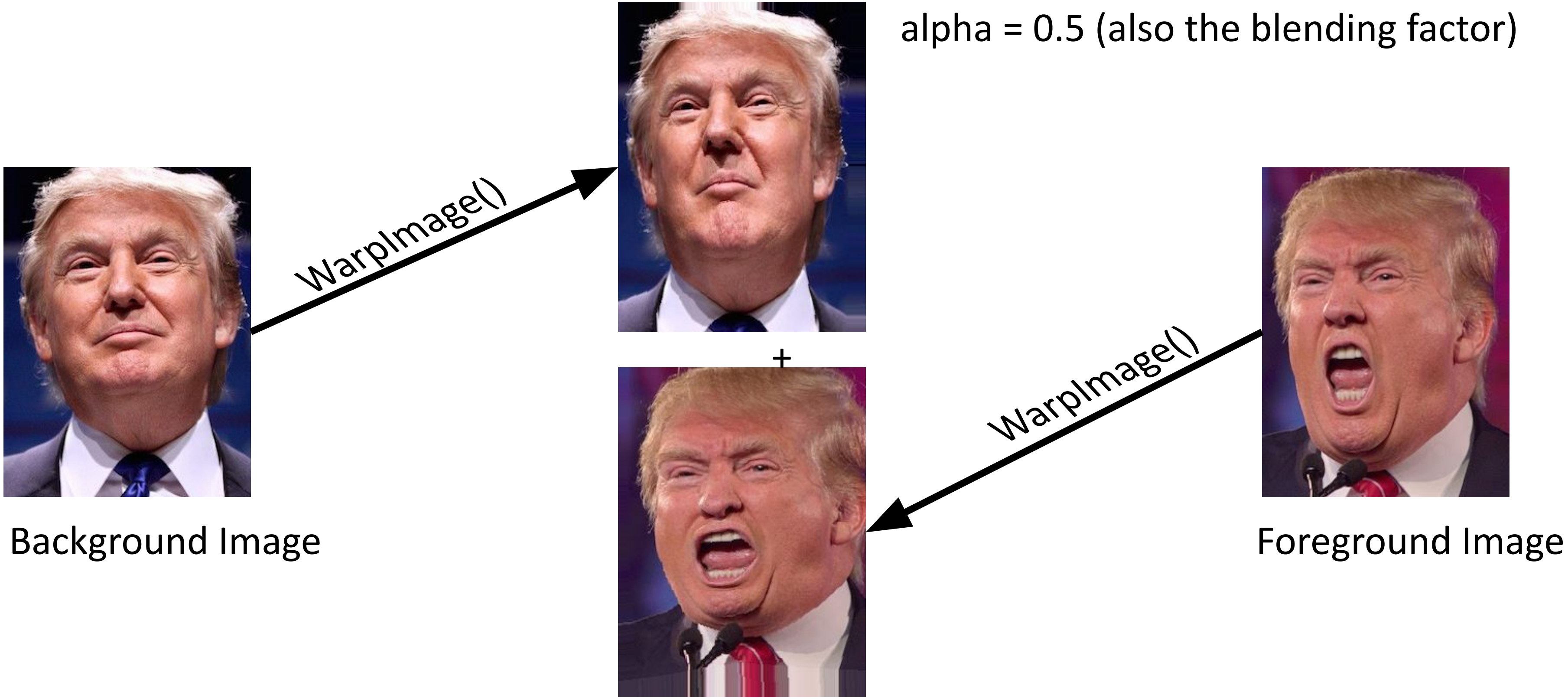



Foreground Image

current_line[i] = (1 – alpha) * background_lines[i] + alpha * foreground_lines[i]

# Blending

alpha = 0.5 (also the blending factor)



# Blending

alpha = 0.5 (also the blending factor)

Background Image

Foreground Image