Finishing Up Assignment 1: Image Processing

COS 426: Computer Graphics (Fall 2022)

Guðni Gunnarsson, Yuanqiao Lin, Yuting Yang

Picking up where we left off last week...

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

Color
- Grayscale
- Saturation
- White balance
- Histogram matching

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

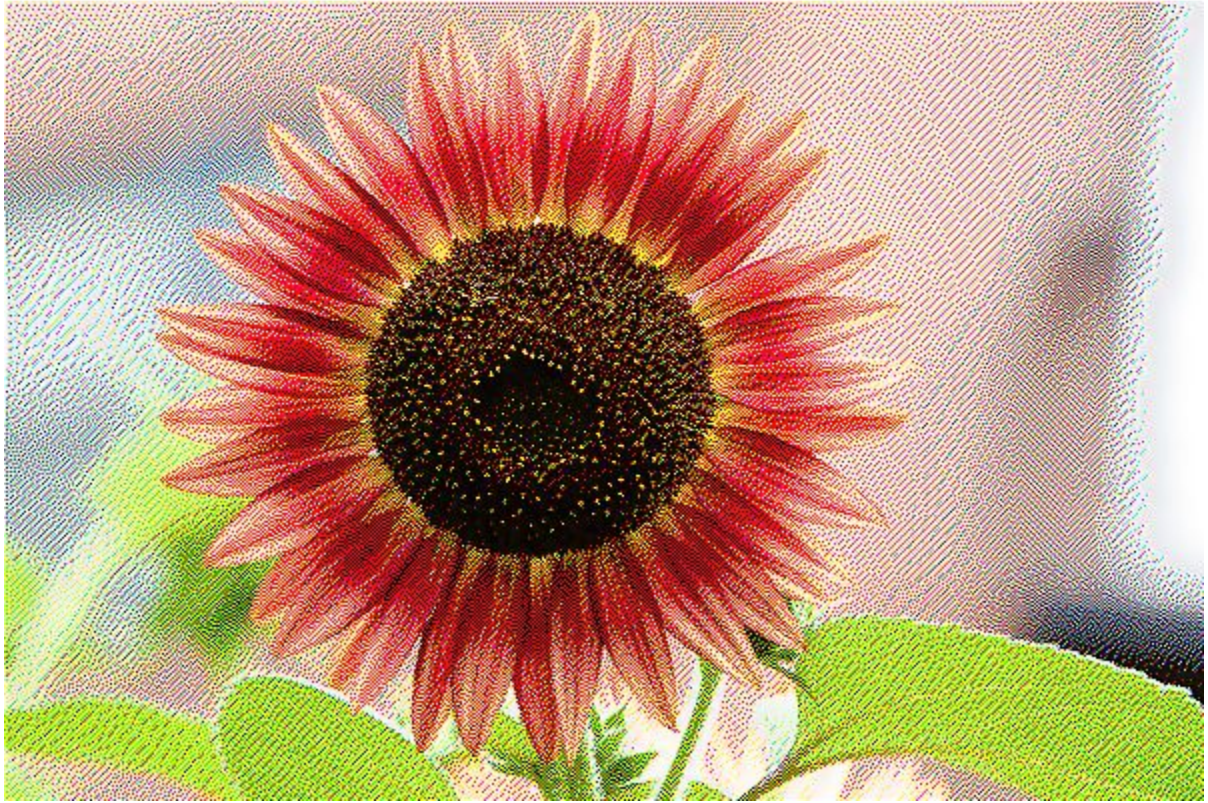
Dithering
- Quantization
- Random dithering
- Floyd-Steinberg error diffusion
- Ordered dithering

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

Composite
- Composite
- Morph

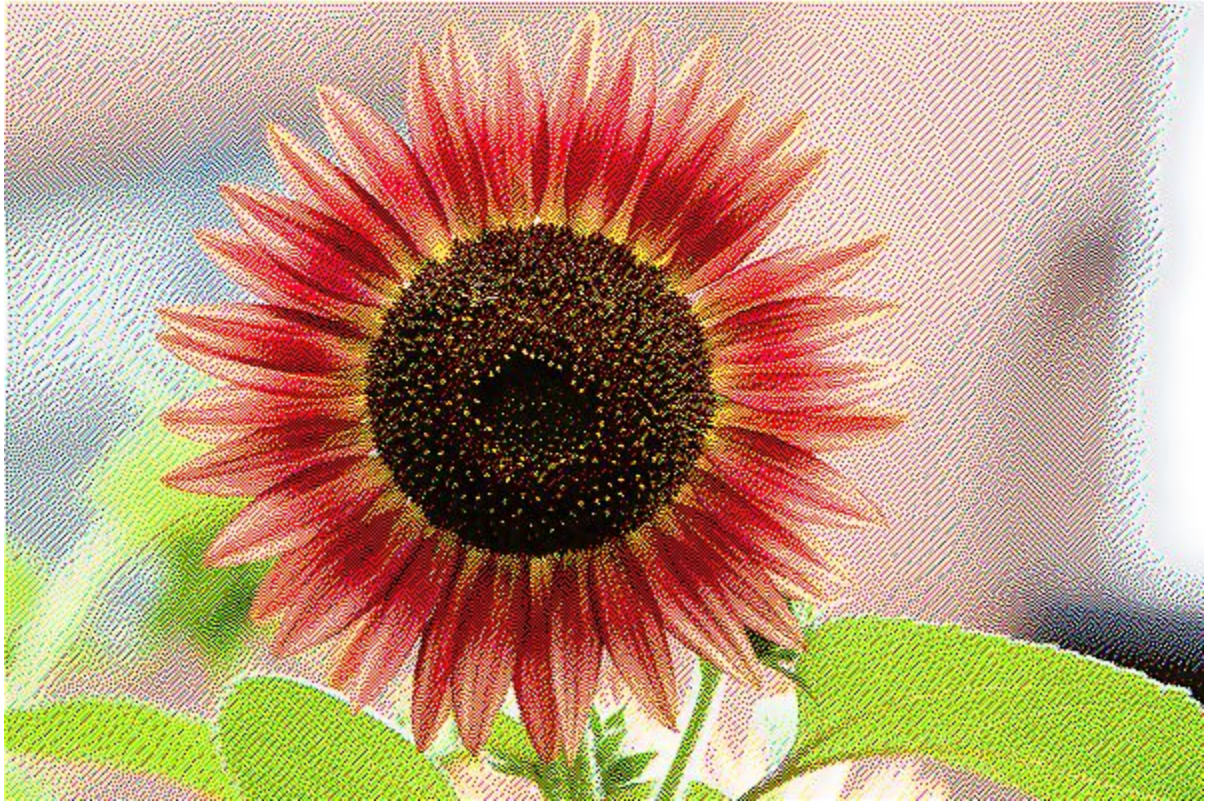
This week's precept will focus specifically on this topic

A Familiar Pattern



Notice anything familiar about the pattern?

Why Dither?

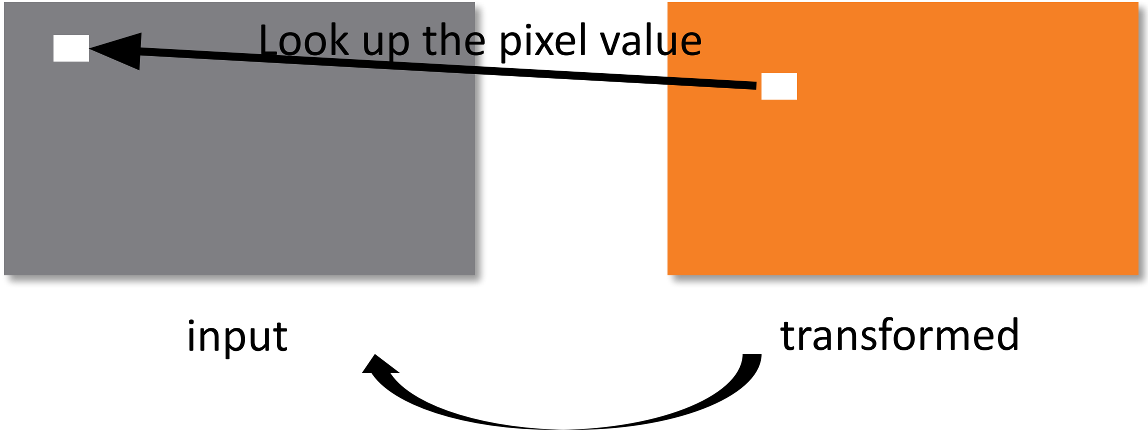


It’s a Floyd-Steinberg dither over RGB channels (1 bit each)!

This filter was often used to compress web GIFs — look for the artifact in old-school animations!

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

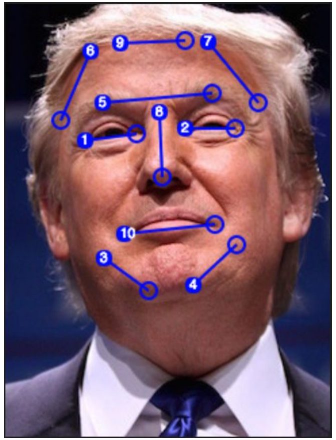
foregroundImg(alpha channel)

Result

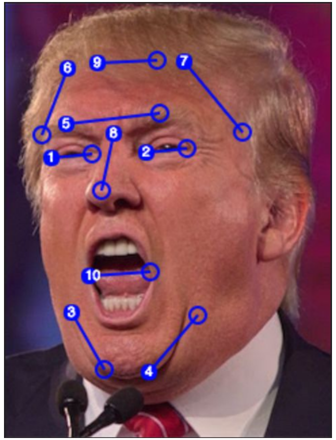
Morph

- 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 correspondence
  - warp the foreground image to the intermediate correspondence
  - blend using alpha

Interpolate Morph Lines



Background Image



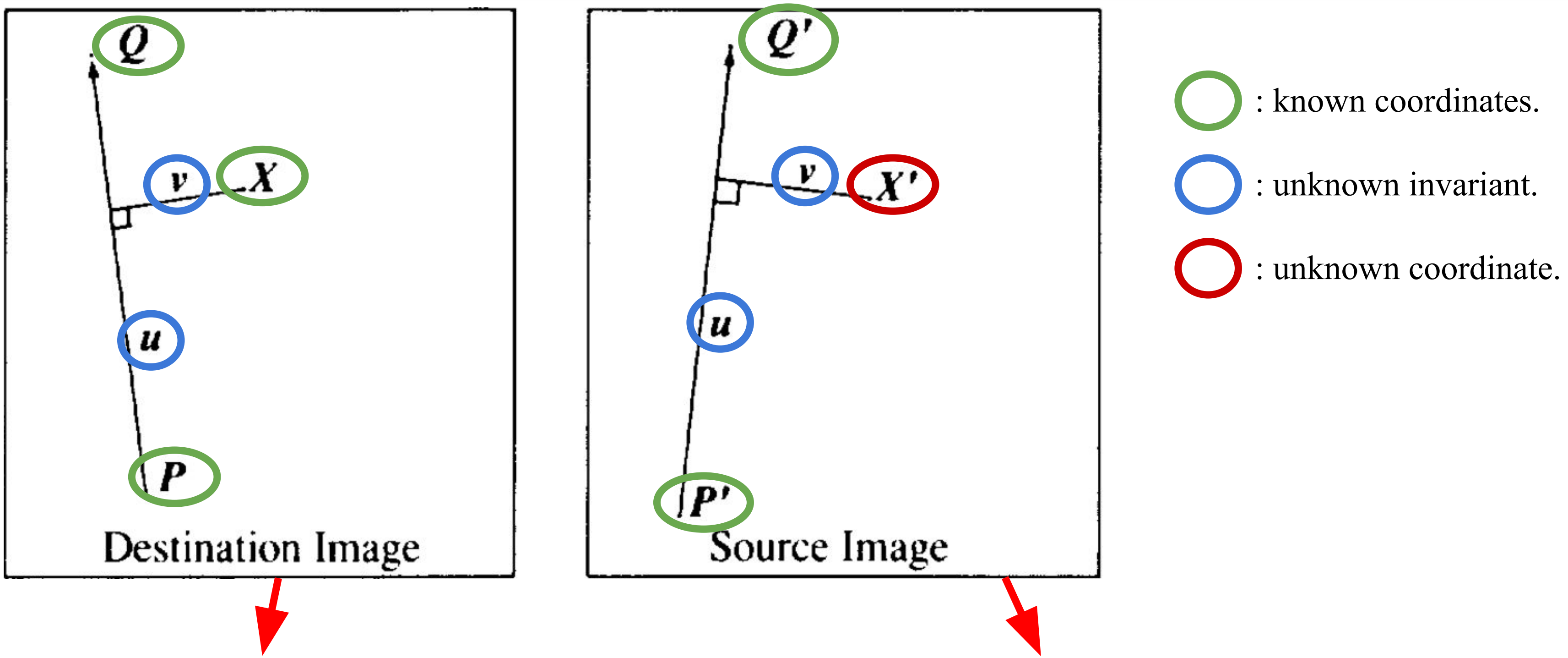
Foreground Image

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

Morph Algorithm Overview

1. Warp for a single line pair
2. Warp for many line pairs
3. For a fixed t, define the current line pairs as an interpolation between initial and final lines
4. Warp initial image I to **intermediate** I' and final image F to **intermediate** F' using current line pairs from Step 3
5. Alpha blend I' and F' using t
6. Vary t to get a morphing animation

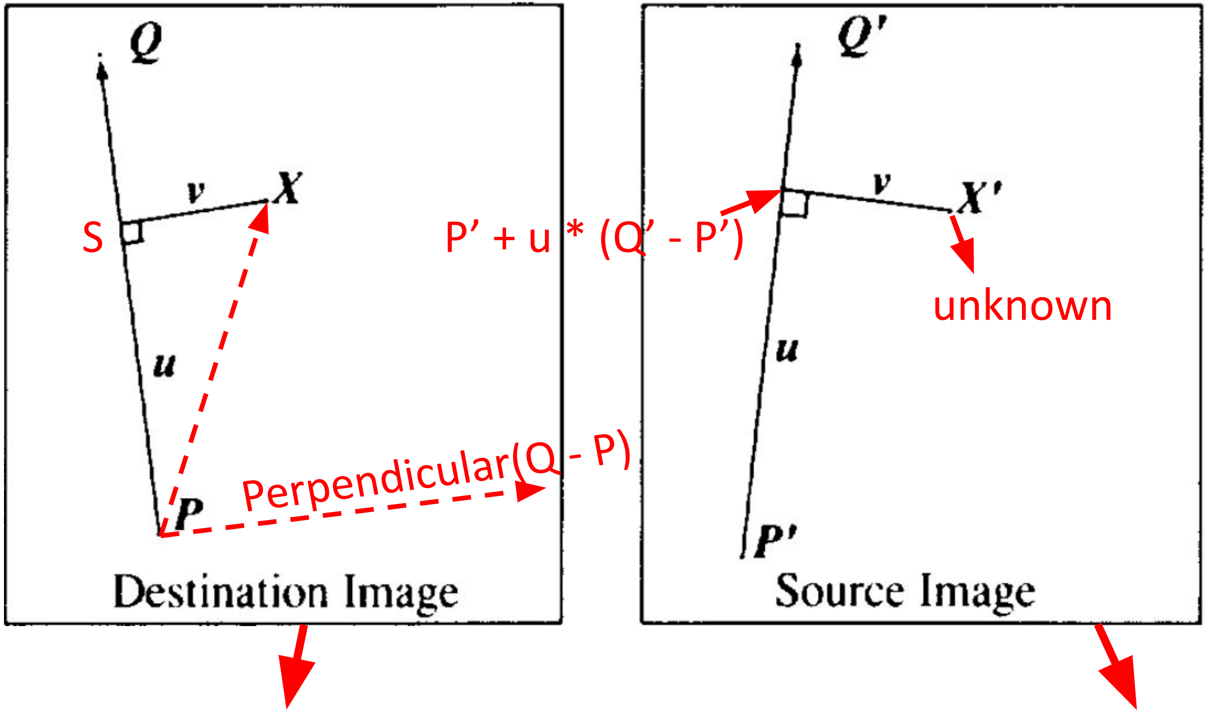
Warp Image (Single Line)



Warped background or foreground (currently undefined)

Pixel source (background or foreground)

Warp Image (Single Line)



Let S be the projection point of X onto PQ

u = fraction of SP’s signed length over PQ’s absolute length

v = X’s signed distance to PQ, or to say, signed length of SX

Warped background or foreground (currently undefined)

Pixel source (background or foreground)

Warp Image (Single Line)

scalar

- $u = \frac{(X-P)\cdot(Q-P)}{||Q-P||^2}$ = Projection of PX onto PQ

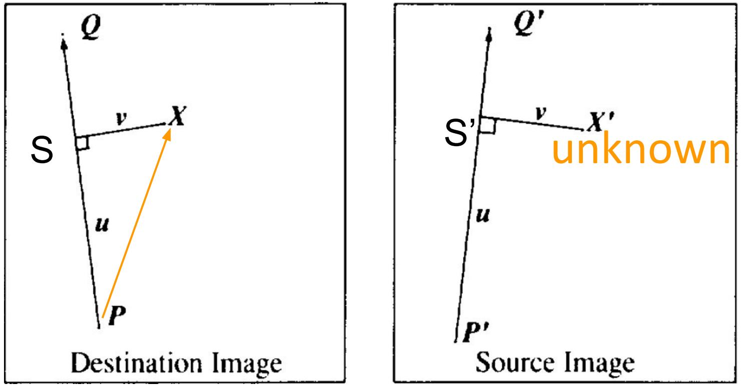
scalar

- $v = (X-P)\cdot\boxed{\frac{Perpendicular(Q-P)}{||Q-P||}}$ unit vector

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

- $X' = P' + u\cdot(Q'-P') + v\cdot\boxed{\frac{Perpendicular(Q'-P')}{||Q'-P'||}}$ unit vector

S’ = P’ + u * (Q’ - P’)



Want to map X in destination image to unknown pixel X’ in source image which contains current line

Warp Image (Single Line)

scalar
- $u = \frac{(X-P)\cdot(Q-P)}{||Q-P||^2}$ = Projection of PX onto PQ

scalar
- $v = \frac{(X-P)\cdot \boxed{Perpendicular(Q-P)}}{\boxed{||Q-P||}}$ unit vector

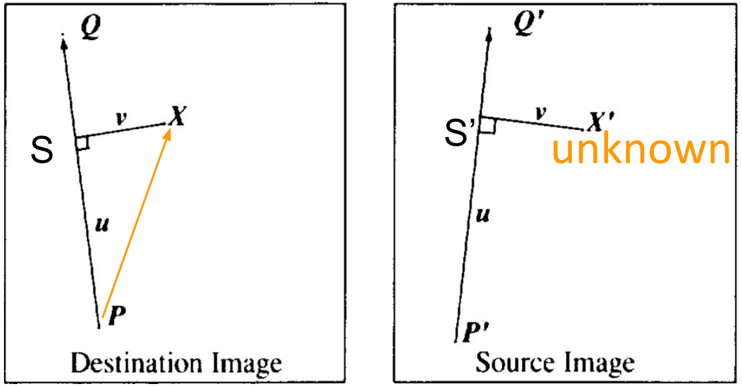
If Q - P = (x, y),
Perpendicular(Q – P) = (y, -x)

- $X' = P' + u\cdot(Q' - P') + \frac{v\cdot \boxed{Perpendicular(Q'-P')}}{\boxed{||Q'-P'||}}$ unit vector

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

Contribution (weight) of line segment PQ to the warping of X's location
Each line segment contributes some weight

S' = P' + u * (Q' - P')

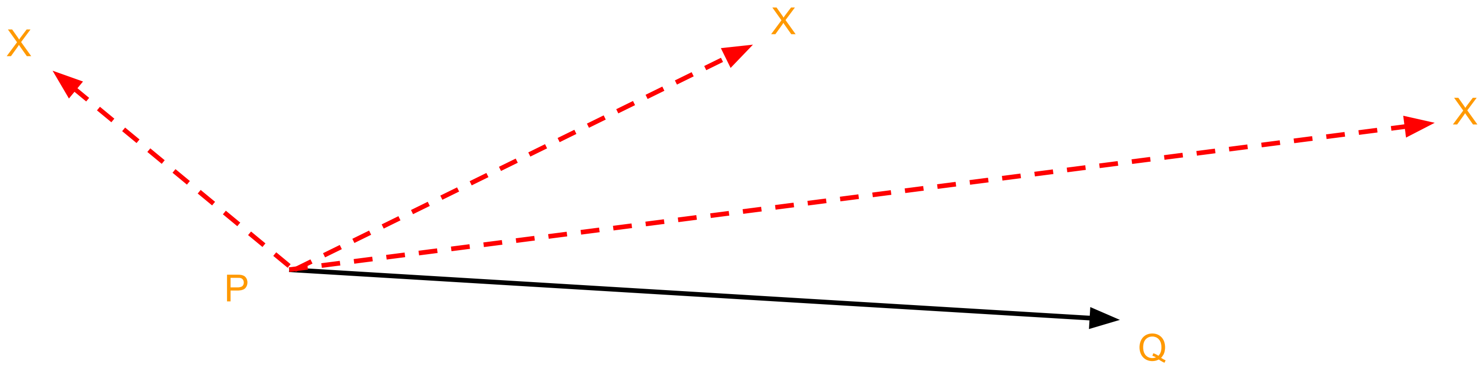


Want to map X in destination image to unknown pixel X' in source image which contains current line

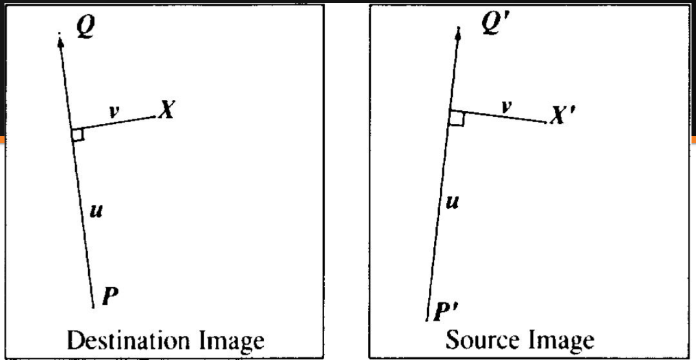
Warp Image (Single Line)

$$dist = shortest\ distance\ from\ X\ to\ PQ$$

- 0 <= u <= 1: dist = |v|
- u < 0: dist = ||X – P||
- u > 1: dist = ||X – Q||



Warp Image (Many Lines)

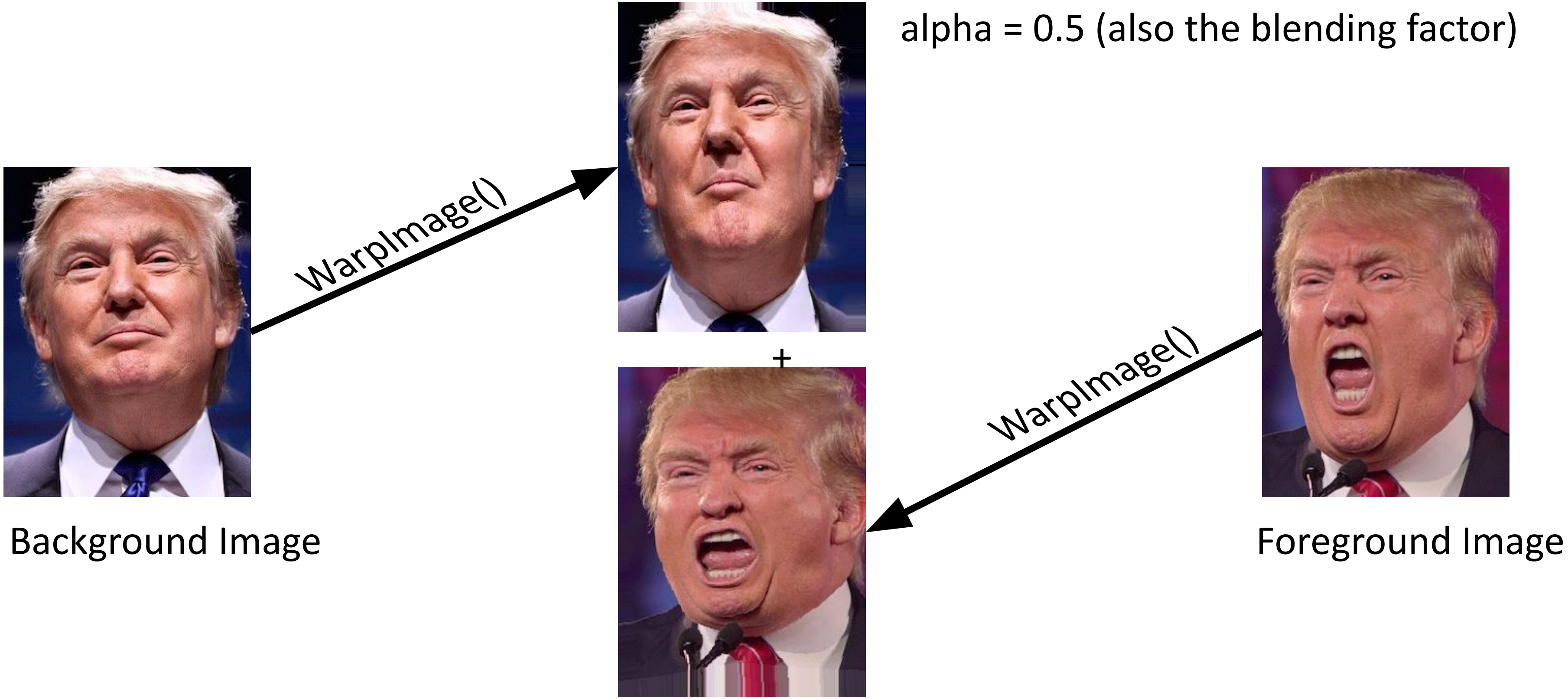


For each pixel X in the destination
- $DSUM = (0,0)$
- $weightsum = 0$ Track total weight for later averaging
- For each line $P_i Q_i$
  - calculate u,v based on $P_i Q_i$
  - calculate X'_i based on u,v and $P_i'Q_i'$
  - calculate displacement $D_i = X_i' - X_i$ for this line
  - $dist$ = shortest distance from X to $P_i Q_i$
  - $weight = (length^p / (a + dist))^b$
  - $DSUM \mathrel{+}= D_i * weight$
  - $weightsum \mathrel{+}= weight$
- $X' = X + DSUM / weightsum$ Repeat for all lines and then average based on weight
- destinationImage(X) = sourceImage(X')

Algorithm described before for a single line

Blending

alpha = 0.5 (also the blending factor)



Blending

Vary this alpha to get an animation

alpha = 0.5 (also the blending factor)

Background Image

Foreground Image

Morph Algorithm Sketch

```
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
```

Q&A

Course Logistics Update

- New course website incoming!
  - Preview at https://reillybova.github.io/COS426-Website/
  - Should have everything, but may be slightly buggy as we work out kinks
  - If you notice any problems, please make a public Piazza post under the "website" folder
- Web Framework specs (for those interested):
  - ReactJS for state-based logic and modularity
  - MaterialUI to build a Material Design compliant interface
  - GatsbyJS to compile the React App to static server files (allows us to host site as a normal webpage, and makes it blazing fast)
  - Content generate from Markdown

Fill out the Assignment 0 Feedback Form

Do this **now** — it takes less than a minute:

- https://forms.gle/o2ea1iJ978zY6Kd78

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

An Update on the Bilateral Filter

- Compute color distance in RGB space, scaled to [0, 255].

Spatial distance component

Color distance component

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