Finishing Up Assignment 1: Image Processing
Picking up where we left off last week...

<table>
<thead>
<tr>
<th>Luminance</th>
<th>Dithering</th>
<th>Composite</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brightness</td>
<td>Quantization</td>
<td>Composite</td>
</tr>
<tr>
<td>Contrast</td>
<td>Random dithering</td>
<td></td>
</tr>
<tr>
<td>Gamma</td>
<td>Floyd-Steinberg error diffusion</td>
<td></td>
</tr>
<tr>
<td>Vignette</td>
<td>Ordered dithering</td>
<td></td>
</tr>
<tr>
<td>Histogram equalization</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Color</td>
<td>Resampling</td>
<td></td>
</tr>
<tr>
<td>Grayscale</td>
<td>Bilinear sampling</td>
<td></td>
</tr>
<tr>
<td>Saturation</td>
<td>Gaussian sampling</td>
<td></td>
</tr>
<tr>
<td>White balance</td>
<td>Translate</td>
<td></td>
</tr>
<tr>
<td>Histogram matching</td>
<td>Scale</td>
<td></td>
</tr>
<tr>
<td>Filter</td>
<td>Rotate</td>
<td></td>
</tr>
<tr>
<td>Gaussian</td>
<td>Swirl</td>
<td></td>
</tr>
<tr>
<td>Sharpen</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Edge detect</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Median</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bilateral filter</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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!

Look up the pixel value
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 $x' = x \times sx$, $y' = y \times sy$:
  $$I'(x', y') = I(x' / sx, y' / sy)$$
Composite

\[ \text{output} = \alpha \times \text{foreground} + (1 - \alpha) \times \text{background} \]

\( \alpha \) is the alpha channel foreground
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
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)

- Green circle: known coordinates.
- Blue circle: unknown invariant.
- Red circle: unknown coordinate.
Warp Image (Single Line)

Warped background or foreground (currently undefined)

Let $S$ be the projection point of $X$ onto $PQ$

$u = \text{fraction of } SP's \text{ signed length over } PQ's \text{ absolute length}$

$v = X's \text{ signed distance to } PQ, \text{ or to say, signed length of } SX$

$P' + u \cdot (Q' - P')$
Warp Image (Single Line)

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

scalar
• $v = \frac{(X-P) \cdot \text{Perpendicular}(Q-P)}{||Q-P||}$

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

If $Q - P = (x, y)$, $\text{Perpendicular}(Q - P) = (y, -x)$

unit vector
• $S' = P' + u \cdot (Q' - P')$

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

- **\( X' = P' + u \cdot (Q' - P') + \frac{v \cdot \text{Perpendicular}(Q'-P')}{||Q'-P'||} \)** 

- **dist = shortest distance from X to PQ**
  - \( 0 \leq u \leq 1 \): dist = |v|
  - \( u < 0 \): dist = ||X - P||
  - \( u > 1 \): dist = ||X - Q||

- **weight = \( \left(\frac{\text{length}^p}{a + \text{dist}}\right)b \)**
  - 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

If \( Q - P = (x, y) \), \( \text{Perpendicular}(Q - P) = (y, -x) \)

\[ S' = P' + u \cdot (Q' - P') \]

Want to map X in destination image to unknown pixel X’ in source image which contains current line
Warp Image (Single Line)

$\text{dist} = \text{shortest distance from } X \text{ to } PQ$

- $0 \leq u \leq 1$: $\text{dist} = |v|$
- $u < 0$: $\text{dist} = ||X - P||$
- $u > 1$: $\text{dist} = ||X - Q||$
For each pixel \( X \) in the destination
\[ DSUM = (0,0) \]
\[ weightsum = 0 \]
Track total weight for later averaging

For each line \( P_iQ_i \)
- calculate \( u,v \) based on \( P_iQ_i \)
- calculate \( X’_i \) based on \( u,v \) and \( P_iQ_i’ \)
- calculate displacement \( D_i = X_i’ - X_i \) for this line
- \( dist = \) shortest distance from \( X \) to \( P_iQ_i \)
- \( weight = (length^p / (a + dist))^b \)

\[ DSUM += D_i \times weight \]
\[ weightsum += weight \]

\[ X' = X + DSUM / weightsum \]

Repeat for all lines and then average based on weight

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)
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
    end

    \( \text{Warp}_0 = \text{WarpImage}(\text{Image}_0, L_0, L) \)
    \( \text{Warp}_1 = \text{WarpImage}(\text{Image}_1, L_1, L) \)

    foreach pixel p in FinalImage do
        Result(p) = (1-t) \( \text{Warp}_0 \) + t \( \text{Warp}_1 \)
    end
end
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 generated from Markdown
Fill out the Assignment 0 Feedback Form

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

- [https://forms.gle/o2ea1iJ978zY6Kd78](https://forms.gle/o2ea1iJ978zY6Kd78)
Ordered dithering

Pseudo code for n-bit case:

\[ i = x \mod m \]
\[ j = y \mod m \]
\[ \text{err} = I(x, y) - \text{floor}_\text{quantize}(I(x, y)) \]
\[ \text{threshold} = \frac{D(i, j) + 1}{m^2 + 1} \]
\[ \text{if } \text{err} > \text{threshold} \]
\[ P(x, y) = \text{ceil}_\text{quantize}(I(x, y)) \]
\[ \text{else} \]
\[ P(x, y) = \text{floor}_\text{quantize}(I(x, y)) \]

- \( \text{floor}_\text{quantize}(p) \)
  \[ = \frac{\text{floor}(p \times (2^n - 1))}{(2^n - 1)} \]
- \( \text{ceil}_\text{quantize}(p) \)
  \[ = \frac{\text{ceil}(p \times (2^n - 1))}{(2^n - 1)} \]

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

n=1 example
An Update on the Bilateral Filter

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

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