Introducing Assignment 1: Image Processing

COS 426: Computer Graphics (Spring 2019)
COS426 Assignment 1
Image Processing — Interactive Mode

Switch to: Writeup

Student Name <NetID>
• Useful functions
  – Push Image
  – Animation: generate gif animation using (min, step, max).
  – MorphLines: specify line correspondences for morphing
  – BatchMode: fix current parameter settings
• Features to implement
  – SetPixels: set pixels to certain colors (A0)
  – Luminance: change pixel luminance
  – Color: remap pixel colors
  – Filter: convolution/box filter
  – Dithering: reduce visual artifacts due to quantization \( \approx \) cheat our eyes
  – Resampling: interpolate pixel colors
  – Composite: blending two images
  – Misc
A few reminders…

• Don’t try to exactly replicate example images.
• Choose parameters which give you best results.
• Have fun!
Changing Contrast

- **GIMP formula**
  - value = (value - 0.5) * (tan ((contrast + 1) * PI/4) ) + 0.5;

- **Notes:**
  - When contrast=1, tan(PI/2) is infinite. Using Math.PI can avoid this issue.
  - Do pixel.clamp() 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)
- argument of $\text{gammaFilter}()$ is $\log(\gamma)$
Vignette

- Pixels within innerR remain unchanged
- Pixels outside outerR are black
- Pixels between innerR and outerR should be multiplied with a value in $[0, 1]$:
  - $\text{Multiplier} = 1 - \frac{(R - \text{innerR})}{(\text{outerR} - \text{innerR})}$
  - $R = \sqrt{x^2 + y^2} / \text{halfdiag}$
Transform an image so that it has flat histogram of luminance values.
Histogram Matching

Transform an image so that it has the same histogram of luminance values as the 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/texture_colour/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: $x' = \frac{x'}{L-1}$
Saturation

- $\text{pixel} = \text{pixel} + (\text{pixel} - \text{gray} (\text{pixel})) \times \text{ratio}$
- Do clamp()
whitebalance(image, rgb\_w)

\[ [L_w, M_w, S_w] = \text{rgb2lms}(rgb_w) \]

for each pixel \( x \) in image

\[ [L, M, S] = \text{rgb2lms}(\text{image}(x)) \]

\[ L = L / L_w \]

\[ M = M / M_w \]

\[ S = S / S_w \]

image\_out(x) = \text{lms2rgb}(L, M, S)

• Hints:
  • Use rgbToXyz(), xyzToLms(), lmsToXyz(), xyzToRgb()
  • Do clamp()
Convolution (Gaussian/Sharpen/Edge)

<table>
<thead>
<tr>
<th>w1</th>
<th>w2</th>
<th>w3</th>
</tr>
</thead>
<tbody>
<tr>
<td>w4</td>
<td>w5</td>
<td>w6</td>
</tr>
<tr>
<td>w7</td>
<td>w8</td>
<td>w9</td>
</tr>
</tbody>
</table>
• Weights can be normalized depending on the application
• Edges? (not required)
  – Mirror boundary
  – Zero padding
  – Use part of the kernel only
Gaussian filter

• Create a new image to work on
• Weights should be normalized, so that they sum to 1.
• Formula:
  \[ G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} \]
  – \( x \) = distance to the center of the kernel
• Speed up:
  – First apply a 1D Gaussian kernel vertically and then a 1D Gaussian kernel horizontally
Edge

• Kernel:

<table>
<thead>
<tr>
<th>-1</th>
<th>-1</th>
<th>-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>8</td>
<td>-1</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>

Inside boundary

<table>
<thead>
<tr>
<th>3</th>
<th>-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>

At boundary

• Don’t normalize weights
• 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

• Don’t normalize weights
# Edge Filter vs Sharpen Filter

\[
\text{Convolution(Image, Sharpen Filter)} = \text{Convolution(Image, Edge Filter)} + \text{Image}
\]

<table>
<thead>
<tr>
<th>Edge Filter</th>
<th>Sharpen Filter</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>-1</td>
<td>8</td>
</tr>
<tr>
<td>-1</td>
<td>9</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>
Median

• Use a window (similar to convolution)
• Choose the median within the window
• Sorting: sort by RGB separately / sort by luminance

RGB Example
Bilateral

• Combine Gaussian filtering in both spatial domain and color domain
• Weight formula of filter for pixel \((i, j)\):

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

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

- Quantize a pixel within [0, 1] using n bits
  - \[\text{round}(p \times (2^n-1)) / (2^n-1)\]

n=1 example
Random dithering

• Before quantization:
  - $p = p + \frac{\text{random()} - 0.5}{(2^n-1)}$
  - $n$ is number of bits per channel

$n=1$ 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
Ordered dithering

Pseudo code for n-bit case:

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

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

n=1 example
Resampling

• Bilinear interpolation

\[
f(x, y) = \frac{1}{(x_2 - x_1)(y_2 - y_1)} \left( f(Q_{11})(x_2 - x)(y_2 - y) + f(Q_{21})(x - x_1)(y_2 - y) \\
+ f(Q_{12})(x_2 - x)(y - y_1) + f(Q_{22})(x - x_1)(y - y_1) \right)
\]

(from wikipedia)
Resampling

- Gaussian interpolation
  - Weights:
    \[ G(d, \sigma) = e^{-d^2/(2\sigma^2)} \]
  - Weights need to be normalized, so that sum up to 1
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 \( x' = x * sx, \ y' = y * sy \):
  \[
  I'(x', y') = I(x' / sx, y' / sy)
  \]
• output = alpha * foreground + (1 - alpha) * background
• alpha is the alpha channel foreground
• 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
GenerateAnimation(Image_0, L_0[...], Image_1, L_1[...])
begin
    foreach intermediate frame time t do
        for i = 1 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

- \( u = \frac{(X-P) \cdot (Q-P)}{||Q-P||^2} \)
- \( v = \frac{(X-P) \cdot \text{Perpendicular}(Q-P)}{||Q-P||} \)  \( \text{unit vector} \)
- \( X' = P' + u \cdot (Q' - P') + \frac{v \cdot \text{Perpendicular}(Q' - P')}{||Q'-P'||} \)  \( \text{unit vector} \)

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

- \( \text{weight} = \left( \frac{\text{length}^p}{a + \text{dist}} \right)^b \)
  - we use \( p = 0.5, a = 0.01, b = 2 \)

Contribution of this line segment \( PQ \) to the warping of \( X \)'s location
If \( Q - P = (x, y) \),

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

\[
P' + u \cdot (Q' - P')
\]
Warp Image

For each pixel $X$ in the destination

$DSUM = (0, 0)$

$weightsum = 0$

For each line $P_iQ_i$

- calculate $u, v$ based on $P_iQ_i$
- 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 $P_iQ_i$
- $weight = (length^p / (a + dist))^b$
- $DSUM += D_i \cdot weight$
- $weightsum += weight$

$X' = X + DSUM / weightsum$

$destinationImage(X) = sourceImage(X')$
current_line[i] = (1 – alpha) * background_lines[i] + alpha * foreground_lines[i]
Blending

alpha = 0.5 (also the blending factor)

Background Image

WarpImage()

+ alpha = 0.5 (also the blending factor)

WarpImage()

Foreground Image
alpha = 0.5 (also the blending factor)
Q&A