COS426 Precept2

Image Processing

Presented by: Linguang Zhang
Assignment structure
GUI

COS426 Assignment 1
Image Processing — Interactive Mode

Switch to: Writeup

Student Name <NetID>
GUI

• 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
  • Luminance: change pixel colors
  • Color: remap pixel colors
  • Filter: convolution/box filter
  • Dithering: ≈ 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)
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]:
  - Multiplier = 1 - (R - innerR) / (outerR - innerR)
  - \( R = \sqrt{x^2 + y^2} / \text{halfdiag} \)
Histogram Equalization

Before

After
Histogram Matching

reference image: town

reference image: flower

reference image: town

reference image: flower
Histogram Equalization/Matching

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

(source: http://paulbourke.nettexture_colour/equalisation/)
Histogram Equalization/Matching

• Target cdf:
  • Equalization:
    • $cdf(i) = \frac{i}{L-1}$
  • Matching:
    • $cdf$ of the reference image

• Implementation
  • Equalization
    • $x' = cdf(x)$
  • Matching
    • $x' = \operatorname{argmin}_i |cdf(x) - cdf_{ref}(i)|$
    • Convert back: $x' = \frac{x'}{L-1}$
Saturation

- pixel = pixel + (pixel - gray(pixel)) * ratio
- Do clamp()
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)
Convolution (Gaussian/Sharpen/Edge)

• 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
• 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:
  • Apply 1D kernel vertically and horizontally
Edge

- Kernel:

```
-1 -1 -1
-1  8 -1
-1 -1 -1
```

- Don’t normalize weights
- Optional to invert the edge map: pixel = 1 - pixel
Sharpen

• Kernel:

\[
\begin{array}{ccc}
-1 & -1 & -1 \\
-1 & 9  & -1 \\
-1 & -1 & -1 \\
\end{array}
\]

• Don’t normalize weights
Median

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

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

• Do not penalize color difference, otherwise Gaussian smoothing

• Scaling sigmaR
  • sigmaR’ = sigmaR \times factor
  • factor = win_size \times \sqrt{2} in the reference solution
  • win_size = 2 \times winR + 1
Quantization

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

• Before quantization:
  • $p = p + (\text{random()} - 0.5)/(2^n-1)$
Ordered dithering

Pseudo code:

\[
\begin{align*}
i &= x \mod n \\
j &= y \mod n \\
err &= I(x, y) - \text{quantize}(I(x, y)) \\
\text{threshold} &= D(i, j) / (n^2 + 1) \\
\text{if } err > \text{threshold} & \quad \text{then} \\
\quad P(x, y) &= \text{ceil}(I(x, y)) \\
\text{else} & \quad \text{false} \\
\quad P(x, y) &= \text{floor}(I(x, y))
\end{align*}
\]

Here quantize() uses floor()

\[
\begin{bmatrix}
1 & 9 & 3 & 11 \\
13 & 5 & 15 & 7 \\
4 & 12 & 2 & 10 \\
16 & 8 & 14 & 6
\end{bmatrix}
\]

n = 4
Floyd-Steinberg error diffusion

- Spread quantization error over neighboring pixels
- Results look more natural
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

(Values in the above matrix are just examples)
Transformation (scale/rotate/swirl)

Try to guess the formula from the behavior of swirl 😊

• Inverse mapping

Look up the pixel value

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

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

Can be obtained using the GUI
Morph

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

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

\[
dist = \text{shortest distance from } X \text{ to } P_i Q_i
\]

\[
weight = \frac{\text{length}^p}{(a + dist)^b}
\]

\[
DSUM += D_i \ast weight
\]

\[
weightsum += weight
\]

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

\( \text{destinationImage}(X) = \text{sourceImage}(X') \)
Warp Image

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