digital image processing

- avoid/correct errors
- restore
- enhance
- analyze
- create

types of techniques

- simple pixel modification
- interpolation/extrapolation
- compositing
- convolution
- dithering
- warping
- morphing
- misc. effects

simple pixel modification

\[ f: [0,1]^3 \rightarrow [0,1]^3 \]

apply function \( f \) to each pixel of input image
simple pixel modification

• convert to gray
• threshold
• invert
• brighten/darken

convert to gray

\[ f(r,g,b) = c_r \cdot r + c_g \cdot g + c_b \cdot b \]

\[ c_r = .2126, \quad c_g = .7152, \quad c_b = .0722 \]

Implementation Details

Our image processor will read uncompressed BMP files that contain
- 8 bit images: i.e. 1 channel, 8 bit per pixel per channel (grayscale)
- 24 bit images: i.e. 3 channel, 8 bit per pixel per channel (RGB)

Implementation Details

Our image class handles 1 and 3 channel images with 1 to 8 bits per pixel per channel.

Your image processing routines should handle all of these cases.
**convert to gray pseudo code**

// I is 3 channel, 8 bits per pixel per channel image
Image* convert_to_gray(image I)

- create new one channel image I' that has the
  same dimensions and bits per pixel per
  channel as I
- for each pixel (i,j)
  - get (r,g,b) values of pixel (i,j) in I
  - set channel 0 of I' to c_r * r + c_g * g + c_b * b
- return ptr to I'

// I is 3 channel, n bits per pixel per channel image
Image* convert_to_gray(image I)

- create new one channel image I' that has the
  same dimensions and bits per pixel per
  channel as I
- for each pixel (i,j)
  - get (r,g,b) values of pixel (i,j) in I
  - set channel 0 of I' to c_r * r + c_g * g + c_b * b
- return ptr to I'

**color values**

n bits per pixel per channel

1. 0, 1, 2, ..., 2^n - 1
2. 0, 1/(2^n-1), 2/(2^n-1), ... , 1

we use this convention
convert to gray pseudo code

Image* convert_to_gray(image I)

• if I is a one channel image return NULL
• create new one channel image I' that has the
  same dimensions and bits per pixel per
  channel as I
• for each pixel (i,j)
  • get (r,g,b) values of pixel (i,j) in I
  • set channel 0 of I' to \( r + g + b \)
• return ptr to I'

threshold

for a given threshold T:

\[ f_T(r,g,b)=(h_T(r),h_T(g),h_T(b)) \]

where

- \( h_T(v)=1 \) if \( v>T \)
- \( h_T(v)=0 \) otherwise

threshold (1 channel)

From here on I'll describe the algorithms for
an input that is 3 channel, 8 bits per pixel per channel
UNLESS the generalizations are not obvious (to me).
invert

\[ f(r,g,b) = (1-r,1-g,1-b) \]

invert (3 channel)

brighten/darken

\[ f_{\alpha}(r,g,b) = (\alpha r, \alpha g, \alpha b) \]

clamp to \([0,1]\)

brighten/darken

for brighten/darken factor \( \alpha > 0 \):

\[ f_{\alpha}(r,g,b) = (\alpha r, \alpha g, \alpha b) \]

clamp to \([0,1]\)

types of techniques

- simple pixel modification
- interpolation/extrapolation
- compositing
- convolution
- dithering
- warping
- morphing
- non-photo-realistic effects

interpolation/extrapolation

Note: all images have same dimensions, channels, bits per pixel
interpolation ($\alpha$ in [0,1])

input image $I_0$

output image: $\alpha I_0 + (1-\alpha)I_1$

interpolation blends $I_0$ and $I_1$

extrapolation ($\alpha > 1$)

input image $I_0$

output image: $\alpha I_0 + (1-\alpha)I_1$

extrapolation (with $\alpha > 1$)
creates an image that moves $I_0$ away from $I_1$

extrapolation ($\alpha < 0$)

input image $I_0$

output image: $\alpha I_0 + (1-\alpha)I_1$

extrapolation (with $\alpha < 0$)
creates an image that moves $I_1$ away from $I_0$

interpolation/extrapolation

brighten/darken

interpolate/extrapolate image with black image

change saturation

interpolate/extrapolate image with black image
interpolation ($\alpha$ in [0,1])

$$\text{output image: } \alpha I_0 + (1-\alpha)I_1$$

interpolation blends $I_0$ and $I_1$

extrapolation ($\alpha > 1$)

$$\text{output image: } \alpha I_0 + (1-\alpha)I_1$$

extrapolation (with $\alpha > 1$) creates an image that moves $I_0$ away from $I_1$

derease saturation

increase saturation

change saturation

interpolate/extrapolate image with

interpolation ($\alpha$ in [0,1])

$$\text{output image: } \alpha I_0 + (1-\alpha)I_1$$

interpolation blends $I_0$ and $I_1$

derease contrast

increase contrast

change contrast

interpolation ($\alpha$ in [0,1])

$$\text{output image: } \alpha I_0 + (1-\alpha)I_1$$

interpolation blends $I_0$ and $I_1$

derease contrast

increase contrast

extrapolation (with $\alpha > 1$) creates an image that moves $I_0$ away from $I_1$
**change contrast**

interpolate/extrapolate image with

**noisify**

noisify

interpolate/extrapolate image with

**invert**

invert

interpolate/extrapolate image with

**a little computation**

- invert: \( f(v) = 1 - v \)
a little computation

- invert: \( f(v) = 1 - v \)
- interpolate/extrapolate: 
  \[
  f(v) = (1 - \alpha) \cdot v + \alpha \cdot x
  \]

What should \( \alpha \) and \( x \) be?

invert

interpolate/extrapolate with

type of techniques

- simple pixel modification
- interpolation/extrapolation
- compositing
- convolution
- dithering
- warping
- morphing
- misc. effects

compositing

generalization of interpolation in which \( \alpha \) varies depending on pixel location
compositing

typically $\alpha \in [0,1]$ so the array of $\alpha$ values can represented by a single channel image called a mask.

compositing

- simple pixel modification
- interpolation/extrapolation
- compositing
- **convolution**
- dithering
- warping
- morphing
- misc. effects

convolution

$v$ is a weighted sum of $v$ and its neighbors.

$$v' = \sum_{i,j} w_{ij} v_{ij}$$

kernel gives weights

$w_{32}$

$w_{22}$

$w_{12}$

$w_{31}$

$w_{21}$

$w_{11}$

$w_{33}$

$w_{23}$

$w_{13}$

$w_{ij} \cdot v_{ij}$

clamp to $[0,1]$ at boundaries?
**edge detect kernel**

\[
\begin{array}{ccc}
-1 & -1 & -1 \\
-1 & 8 & -1 \\
-1 & -1 & -1 \\
\end{array}
\]

**blur kernel**

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

**blur**

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

3x3 blox blur kernel

**edge detect**

[Google image]
Jaggies

5x5 box blur

nXn box blur

Why is it important that the sum of the weights is 1?

Box, triangle and gaussian blurs

1D convolution

How to construct the kernels for 2D triangle and gaussian blurs?

We'll start with 1D blurs.
1D convolution

1D convolution

1D box blur, n=3

1D box blur

weights for 1D blurs

1D triangle blur weights for n=3
1D triangle blur weights for $n=3$

where should we sample?

for convenience let’s say $w$, is sampled at $i$.

how high is the peak?

we want the weights to sum to 1!

how high is peak? let’s start with something convenient then normalize.

what are $s_1$ and $s_3$?
1D triangle blur weights for \( n=3 \)

\[
\begin{align*}
1 & : w_1 = \frac{1}{4} \\
2 & : w_2 = \frac{1}{2} \\
3 & : w_3 = \frac{1}{4}
\end{align*}
\]

General 1D triangle blur

where do we take the \( n \) samples?

General 1D triangle blur

sample \( w_i \) at \( i=1, 2, \ldots, n \)

General 1D triangle blur

what is height of peak?

General 1D triangle blur

Let \( s_i = i \) for \( 1 \leq i \leq (n+1)/2 \)

\[
\begin{align*}
&= (n+1) - i \quad \text{for} \quad (n+1)/2 < i \leq n
\end{align*}
\]

Let \( B = \sum_{i=1}^{n} s_i \)

Let \( w_i = s_i / B \)
1D triangle blur

\[ w_i = \begin{cases} \frac{4i}{(n+1)^2} & \text{for } 0 \leq i \leq \frac{n+1}{2} \\ \frac{4((n+1)-i)}{(n+1)^2} & \text{for } \frac{n+1}{2} < i \leq n+1 \end{cases} \]

1D \rightarrow 2D

How about applying the 1D filter to each row, then to each column!

box blur - rows

\[
\begin{array}{ccc}
v_{00} & v_{01} & v_{02} \\
v_{10} & v_{11} & v_{12} \\
v_{20} & v_{21} & v_{22}
\end{array}
\]

box blur - column

\[
\begin{array}{ccc}
? & (v_{00} + v_{01} + v_{02})/3 & ? \\
? & (v_{10} + v_{11} + v_{12})/3 & ? \\
? & (v_{20} + v_{21} + v_{22})/3 & ?
\end{array}
\]

3x3 box blur

\[
\begin{array}{ccc}
1/3 & 1/9 & 1/9 \\
1/3 & 1/9 & 1/9 \\
1/3 & 1/9 & 1/9 \\
1/3 & 1/3 & 1/3
\end{array}
\]

separability

\[
\begin{array}{ccc}
w_1 & w_1^2 & w_1w_2 \\
w_2 & w_2^2 & w_2w_3 \\
w_3 & w_3^2 & w_3w_2
\end{array}
\]
### 3x3 Triangle Blur

**VOILA!**

<table>
<thead>
<tr>
<th>1/4</th>
<th>1/16</th>
<th>1/8</th>
<th>1/16</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/2</td>
<td>1/8</td>
<td>1/4</td>
<td>1/8</td>
</tr>
<tr>
<td>1/4</td>
<td>1/16</td>
<td>1/8</td>
<td>1/16</td>
</tr>
</tbody>
</table>

### Summary: nxn Triangle Blur

- Compute 1D triangle blur for \( n \) (\( n \) is odd)
  - Compute \( s_i \) for \( i-1, \ldots, n \)
  - Compute normalizing factor \( B = \Sigma s_i \)

\[
\begin{align*}
B &= \Sigma_{i=1}^{n+1} s_i \\
    &= \Sigma_{i=0}^{(n+1)/2} i + \Sigma_{i=(n+1)/2}^{n+1} (n+1)-i \\
    &= (n+1)^2/4 
\end{align*}
\]

### nxn Triangle Blur

- Compute 1D triangle blur for \( n \) (\( n \) is odd)
  - Compute \( s_i \) for \( i-1, \ldots, n \)
  - Compute normalizing factor \( B = \Sigma s_i \)

\[
\begin{align*}
S_i &= i & \text{for } 0 < i \leq (n+1)/2 \\
S_i &= (n+1)-i & \text{for } (n+1)/2 < i \leq n+1 
\end{align*}
\]

### nxn Triangle Blur

- Compute 1D triangle blur for \( n \) (\( n \) is odd)
  - Compute \( s_i \) for \( i-1, \ldots, n \)
  - Compute normalizing factor \( B = \Sigma s_i \)

\[
\begin{align*}
B &= \Sigma_{i=1}^{n} s_i \\
    &= \Sigma_{i=0}^{(n+1)/2} i + \Sigma_{i=(n+1)/2}^{n} (n+1)-i \\
    &= (n+1)^2/4 
\end{align*}
\]

### 1D Triangle Blur

\[
\begin{align*}
w_i &= \frac{4i}{(n+1)^2} & \text{for } 0 < i \leq (n+1)/2 \\
     &= \frac{4((n+1)-i)}{(n+1)^2} & \text{for } (n+1)/2 < i \leq n+1 
\end{align*}
\]
separability

\[
\begin{array}{ccc}
W_1^2 & W_1W_2 & W_1W_3 \\
W_2W_1 & W_2^2 & W_2W_3 \\
W_3W_1 & W_3W_2 & W_3^2 \\
\end{array}
\]

\[
W_1 \quad W_2 \quad W_3
\]

example: \( n = 3 \)

\[
w_i = \frac{4i}{(n+1)^2} \quad \text{for} \ 0 < i \leq \frac{n+1}{2}
\]

\[
w_i = 4\left[\frac{(n+1)-i}{n+1}\right] \quad \text{for} \ \frac{n+1}{2} < i \leq n+1
\]

3x3 triangle blur filter

\[
\begin{array}{ccc}
1/4 & 1/8 & 1/16 \\
1/8 & 1/4 & 1/8 \\
1/16 & 1/8 & 1/16 \\
1/4 & 1/2 & 1/4
\end{array}
\]

box, triangle and gaussian blurs

gaussian function

\[
f(x) = e^{-(x-\mu)^2/\sigma^2}
\]

\( \sigma \) is an input parameter that controls the width of peak

sampled

\[
s_i = e^{-(i-(n+1)/2)^2/\sigma^2} \quad \text{for} \ i = 1, \ldots, n
\]
normalized

\[
B = \sum_{i=1}^{n} e^{-(i-(n+1)/2)^2/\sigma^2}
\]

is the normalizing constant

\[w_i = s_i / B\]

2D gaussian - use separability

\[
\begin{align*}
&\text{w}_1 \\
&\text{w}_2 \text{w}_1 \\
&\text{w}_3 \text{w}_2 \text{w}_1
\end{align*}
\]

\[
\begin{align*}
&\text{w}_1^2 \\
&\text{w}_2 \text{w}_1^2 \\
&\text{w}_3 \text{w}_2 \text{w}_1^2
\end{align*}
\]

\[
\begin{align*}
&\text{w}_1 \text{w}_2 \text{w}_3
\end{align*}
\]

equation

\[
\text{example: } n=3, \sigma=1
\]

\[
\begin{array}{ccc}
\text{1} & \text{2} & \text{3} \\
0.212 & 0.212 & 0.576
\end{array}
\]

\[
\begin{array}{ccc}
0.212 & 0.122 & 0.045 \\
0.576 & 0.332 & 0.122 \\
0.212 & 0.045 & 0.122
\end{array}
\]

\[
\begin{array}{ccc}
0.212 & 0.576 & 0.212
\end{array}
\]

type of techniques

- simple pixel modification
- interpolation/extrapolation
- compositing
- convolution
- dithering
- warping
- morphing
- misc. effects