0.1 Gaussian Blur

- *Algorithm:* Gaussian Blur (algo. 2)
- *Input:* An image
- *Complexity:* $\mathcal{O}(w_k w_i h_i) + \mathcal{O}(h_k w_i h_i)$, depending on the size of filter kernel and the image
- *Data structure compatibility:* Image
- *Common applications:* Image processing, edge detection, computer vision

Gaussian Blur

Gaussian blur is a type of image blurting filter that uses a Gaussian function to reduces details and noise in images.

Description

Algorithm Description

Gaussian blur, or Gaussian smoothing, uses a Gaussian function to calculating the transformation of pixel in an image. Gaussian blur is widely used to graphics or image processing software. Gaussian blur serves as a pre-processing stage for lots of computer vision algorithms in an attempt to enhance image structures.

The formula of an 1-D Gaussian function is expressed as

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$

A 2-D Gaussian function is the product of two 1-D Gaussian function, expressed as [3]

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

where x is the distance from the point (or the origin) in horizon. y is the distance from the point (or origin) in vertex. σ is the standard deviation of the Gaussian distribution. The idea of Gaussian blurring is implemented by convolution. Such convolution is performed by convolving with 1-D Gaussian function first in the horizonal, then the vertical direction.

When calculating a discrete approximation of the Gaussian function, pixels that have a distance of more than three standard deviations can be consider effectively zero [1] since they have little influence. Therefore, contribution from pixels farther than 3σ can be ignored. A filter kernel of a matrix with $\lceil 6\sigma \rceil \times \lceil 6\sigma \rceil$ is enough.

The overall running time is determined by size of the filter kernel and the image itself

$$T = \mathcal{O}(w_k w_i h_i) + \mathcal{O}(h_k w_i h_i)$$

where w_k, h_k is the width and height of the filter kernel, w_i, h_i is the width and height of the image.

Implementation

The filter kernel is generated by 2-D Gaussian function. The size N of the filter kernel $N \times N$ has to be odd. Take the center element as the origin point, horizontal direction and right as $x - axis$ and positive direction, vertical direction and up as $y - axis$ and positive direction. All the value is calculated through this coordinate system. The sum of all the value in a filter kernel should be exactly 1.[[2]] In this algorithm, the filter kernel is a matrix of $\lceil 6\sigma \rceil \times \lceil 6\sigma \rceil$.

Algorithm 1: *getFilterKernel*(σ)

[ht] **Input :** standard deviation σ
Output: a $\lceil 6\sigma \rceil \times \lceil 6\sigma \rceil$ matrix A

for $i = -\lceil 6\sigma \rceil/2$ ***to*** $\lceil 6\sigma \rceil/2$ **do**
    for $j = \lceil 6\sigma \rceil/2$ ***to*** $\lceil 6\sigma \rceil/2$ **do**
        $A[i][j] \leftarrow \frac{1}{2\pi\sigma^2} e^{-\frac{i^2+j^2}{2\sigma^2}}$
    end for
end for
$sum \leftarrow$ the sum of all elements in A
$A \leftarrow A./sum$
return A

Algorithm 2: Gaussian Blur

Input : An image *pix*, standard deviation σ
Output: Blurred image *pix*

```
GaussM = getFilterKernel(σ)
radius ← ⌈6σ⌉/2
                                        /* Processing on the horizonal direction */
for every row of pixels do
                                        /* For the ith row */
    GausSum,rSum,gSum,bSum ← 0
    for ervey column of pixels do
                                        /* For the jth column */
        for k=-randius to radius do
            cur ← j + k
            if 0 ≤ cur ≤ width of the image then
                r,g,b ← rgb value of pix[i][cur]
                rSum ← rSum + r
                gSum ← gSum + g
                bSum ← bSum + b
                GausSum ← GaussM[k + radius]
            end if
        end for
                                        /* The processed rgb value */
        r ← rSum/GausSum
        g ← gSum/GausSum
        b ← bSum/GausSum
        pix[i][j] ← r << 16 | g << 8 | b | 0xff000000
    end for
end for
                                        /* Processing on the vertical direction */
for every column of pixels do
                                        /* For the ith column */
    GausSum,rSum,gSum,bSum ← 0
    for ervey row of pixels do
                                        /* For the jth row */
        for k=-randius to radius do
            cur ← j + k
            if 0 ≤ cur ≤ width of the image then
                r,g,b ← rgb value of pix[i][cur]
                rSum ← rSum + r
                gSum ← gSum + g
                bSum ← bSum + b
                GausSum ← GaussM[k + radius]
            end if
        end for
                                        /* The processed rgb value */
        r ← rSum/GausSum
        g ← gSum/GausSum
        b ← bSum/GausSum
        pix[i][j] ← r << 16 | g << 8 | b | 0xff000000
    end for
end for
return
```

References.

[1] Simon Perkins Bob Fisher and etc. *Spatial Filters - Gaussian Smoothing*. 2006 (cit. on p. 1).

[2] E. Davies. *Machine Vision: Theory, Algorithms and Practicalities*. Academic Press, 1990 (cit. on p. 2).

[3] Richard E. Woods Rafael C. Gonzalez. *Digital Image Processing(Second Edition)*. Prentice Hall, 2003 (cit. on p. 1).