Computer Vision Homework 7

B03902042 宋子維

Description

In this homework, we are asked to generate thinned image of a binary image in *Figure 1-1*. First, downsample the image from 512×512 to 64×64 by using 8×8 block as a unit, and taking the topmost-left pixel as downsample data. Then generate the thinned image.

Note: In this assignment, both mark interior/border and pair relationship operator use 8-connectivity.

Figure 1-1: Binary image.

Programming

I use python to implement the algorithms. There is one python program, namely, **thinning.py**, where I use **pillow** to process basic image I/O. In the program, there are some basic functions and instructions:

1. `PIL.Image.open(img)`: load the image `img` and return a pillow **Image** object.
2. `pix = Image.load()`: return the **PixelAccess** object of **Image** object to `pix`, which offers us to use `pix[x, y]` to access the pixel value at position (x, y).

3. `PIL.Image.new(mode, size)`: create a new pillow **Image** object given its mode and size.

4. `Image.size`: pair (width, height) of **Image** object.

5. `sumTuple((a, b), (c, d))`: return the tuple (a+c, b+d).

6. `product(range(a), range(b))`: return the list of cartesian product of set $\{0, 1, \cdots, a-1\}$ and set $\{0, 1, \cdots, b-1\}$.

Usage

The usage of **thinning.py** is

```
python3 Yokoi.py IMG_IN IMG_OUT FILE_OUT
```

The program will generate thinned image with respect to downsampling image of **IMG_IN** and store the result image into **IMG_OUT**. Since the image after downsampling is very small, the program can also write the result into **FILE_OUT**, where **'*'** means white pixel.

Algorithms

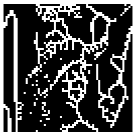

Figure 2-1: Thinned image.

First, we need to downsample the image. Since we take 8×8 block as a unit and take the topmost-left pixel as downsample data, it is clear that the value at (x, y) after downsampling is exactly the the value at $(8x, 8y)$ before downsampling. The following function returns a dictionary object, where key is the position and value is the downsample data corresponding to the key, and the size after downsampling.

```
def downsample(pix, size, BLOCK_LEN):
    # BLOCK_LEN is 8 and size is a 2-tuple (512, 512) in this assignment
    width, height = int(size[0] / BLOCK_LEN), int(size[1] / BLOCK_LEN)

    calPos = lambda x: (x[0]*BLOCK_LEN, x[1]*BLOCK_LEN)
    M = {_: pix[calPos(_)] for _ in product(range(width), range(height))}

    return M, (width, height)
```

Now, take a look at the algorithm of thinning operator. The thinning operator is the composition of three operators: mark interior/border, pair relationship and connected shrink. Moreover, thinning operator keeps doing the operation above until the image doesn't change anymore. Thus, we can get the following code fragment.

```python
def thinning(M, size):
    while True:
        X = mark_interior(M, size)
        Y = pair_rel(X, size)
        ret = connect_shrink(Y, size)

        if ret == M:
            break
        M = ret

    return ret
```

`mark_interior()` is the function that marks the given image with interior or border. **To judge whether the pixel is interior pixel or not, first, it must be white, and second, all of its neighbors must be white**. If the pixel is white but not a interior pixel, then it is border. Otherwise, black. The function `T()` in the following code fragment is to generate the translation at given position with respect to specific kernel, which is 4-connectivity neighbors here.

Note again! I use 8-connectivity!

```python
def mark_interior(M, size):
    width, height = size
    ret = {}
    for cur in product(range(width), range(height)):
        if M[cur] == WHITE:
            trans = T(M, cur, NEIGHBOR)
            if sum(x == WHITE for x in trans.values()) == len(NEIGHBOR):
                ret[cur] = INTERIOR
            else:
                ret[cur] = BORDER
        else:
            ret[cur] = BLACK

    return ret
```

`pair_rel()` is the function that judges whether border pixels are deletable or not. **To do this, simply check whether there exists any interior neighbor of the border pixel or not**. If there is, it is deletable. Otherwise, it's not deletable.

Note again! I use 8-connectivity!

```python
def pair_rel(M, size):
    width, height = size
    ret = {}
    for cur in product(range(width), range(height)):
        if M[cur] == BORDER:
            trans = T(M, cur, NEIGHBOR)
            if sum(x == INTERIOR for x in trans.values()) >= 1:
                ret[cur] = MARKED_BORDER
            else:
                ret[cur] = WHITE
        elif M[cur] == INTERIOR:
            ret[cur] = WHITE
        else:
            ret[cur] = BLACK

    return ret
```

Finally, `connect_shrink()` applies connected shrink operator on given image which deletes marked border pixels without disconnecting regions.

Function `h()` determines whether corner connected and is defined as follows:

$$h(b,c,d,e) = \begin{cases} 1 & \text{if } b = c \text{ and } (d \neq b \text{ or } c \neq b) \\ 0 & \text{otherwise} \end{cases}$$

Function `f()` is defined as follows:

$$f(a_1,a_2,a_3,a_4,x) = \begin{cases} g & \text{if exactly one of } a_1,a_2,a_3,a_4 = 1 \\ x & \text{otherwise} \end{cases}$$

where g means background.

Then output symbol $y = f(a_1,a_2,a_3,a_4,x)$ where

$$\begin{aligned} a_1 &= h(x_0,x_1,x_6,x_2) \\ a_2 &= h(x_0,x_2,x_7,x_3) \\ a_3 &= h(x_0,x_3,x_8,x_4) \\ a_4 &= h(x_0,x_4,x_5,x_1) \end{aligned}$$

Each x_i is defined as follows:

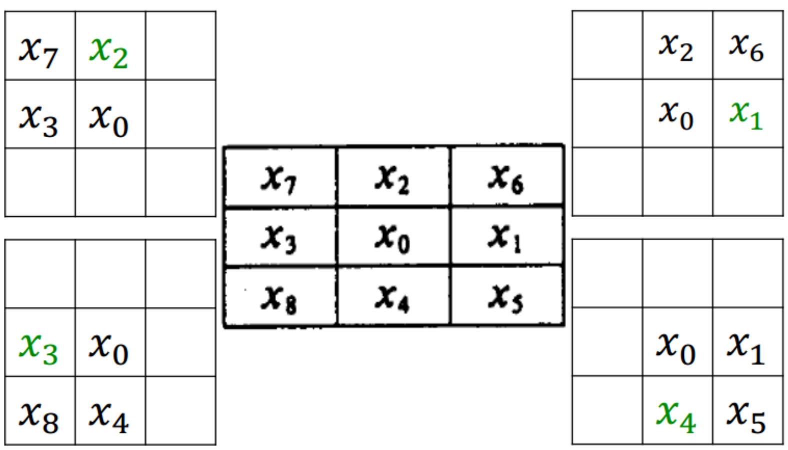

Figure 2-2: 3x3 block and 4 corners of it.

In the following code fragment, `BLOCK` contains 4 corners of 3×3 block above.

```
def h(M, B):
    if (M[B[0]] == M[B[1]]) and (M[B[0]] != M[B[2]] or M[B[0]] != M[B[3]]):
        return 1
    else:
        return 0

def f(a):
    if a[:-1].count(1) == 1:
        return g
    else:
        return a[-1]

def connect_shrink(M, size):
    width, height = size

    for y in range(height):
        for x in range(width):
            cur = (x, y)
            if M[cur] == MARKED_BORDER:
                temp = T(M, cur, _3x3BLOCK)
                trans = {}
                for (k, v) in temp.items():
                    trans[k] = WHITE if v == MARKED_BORDER else v
                M[cur] = f([h(trans, B) for B in BLOCK] + [M[cur]])

    ret = {}
    for y in range(height):
        for x in range(width):
            if M[x, y] in [g, BLACK]:
                ret[x, y] = BLACK
            else:
                ret[x, y] = WHITE

    return ret
```

The following is the result where **'*'** means white pixels.

```
                **          ********     *                  *  *
                 ************ * **  **   ***                  *
                * *  ****  *  *  *****     **               **
   **           * *  ** ** ********         **              *
   **            ** **** **     *  * *       ****          *
  ****           *  *  ** *     *   *         ****        *
 **  *             ** *   *        ****      **  ***
 *   **            **** * ****              **    ***  *
 *   **             *  ***** *              *      *****
**   **                  *   *             **       *
**   **                  *   **            **      **
*    **                  *    **           **      *           *
     **                  *     *           **      *          **
     **                 ****   *           **      *           *
     **         *     ***  **  *            *     **          **
     **         *    ** *   ** **           *      *         **
     **         *    *       ** **          *    ***         **
     **         *             ****** *      *   ****          *
     **        **      *   * ***  *****     *   ***          **
     **        **   * ******** *  *   **    ***** **        **
     **         *    **  *  *  *  *    *******  *  *        **
     **         *    * ***  *  *  *     *  * ** *            *
     **         ****  *  *  * **  *    *** ** *   *         **
     **          *   *** *             * ** *    *          *
     **          *  *  ***             *  *   ***          **
     **          ***   *              **     ***      *     **
     **          *  *  *               *      *       ***** *
     **          *   *      *         **     *       *    ***
     **          *                *  *** *            *     *
     **          *  *            **** ***            **    *
     **           *           * **     *****  *       **   *
     **         * *          ****     ***     *        **  *
     **        ***           ***      * *     *         ** *
     **       *** *        ****   *   *   *   *          ***
     **       **           *** *  *** **  *   *           *
     **     **  *    *     **  *   **  *      *          **
     **      *      **    **   *    ** **     *          **
     **    *              *    *   ******     *          **
     **    *    *     *  **  *******   ***     *        ***
     **    *             *      **      **     *        ***
     **       * *      **       **   *  *      *      *** **
     **                *        **   *  *      *     **     **
     **          *   * *     *  ******* *      *    **      *
     **          *  **       ****     ***      *    *       *
     **   *      * **           *     * *      **   *       *
     **        *  *  *          * *  ***        *** *       *
     **             *           ***   **         *****     ***
     **         *                ***  **         **   **    * *
     **                            ***         **     *    *
      ***      *   **          ***  *          *     **   **
*     *  *     *    *           ** *           *         **
**    * *          *              ***          ***       **
 **   ***          *                *            * *     *
  *   *             * *             *            ***    **
  *   ***    *        *             **           *   ** *
  *   *            ***               *           *    ***
  *   * *                    *        **         ***   **
  **  ***             **               *            *   *
  **  *           *                 **** *         *****
  **  *                            **  ***            *
  **  *             *        *  * **     ** *        **
  **  *                      ******       ***        *
  **  *                      *  *            ***
  **  *         *            ***              *
```