A Research on Guided Thinning Algorithm and Its Implementation by Using C#

Jia Liang *Sanctifier.jia@yahoo.com.cn*
School of Information Science & Engineering
Chang Zhou University
Chang Zhou, 213164, China

Abstract

After the advent of C# in year 2000, it's gradually and widely applied in commercial software developments on various Microsoft platforms. To seamlessly merge image processing algorithms with the technical trend, one efficient means is implementing algorithms directly by using C#. Since there hasn't any way to perform pixel-level operations in the development environment named Visual Studio which integrates C#, a class encapsulating image processing algorithms is developed, and guided thinning algorithm is implemented as a method of the class. The amelioration of the algorithm is made through employing extension methods, manipulating pointers and modifying the condition of executing thinning. Under these conditions, the skeletonization is smoothly implemented and the resulting data is visualized as a skeletonized binary image.

Keywords: Skeletonization, Guided Thinning Algorithm, C#.

1. INTRODUCTION

The skeletonization of binary image is an efficient means assisting image processing algorithms in various applications such as the object detection. In the intermediate-level vision, Hough transformation is one main means to detect objects. For the real-time detections, the performance of a certain Hough transformation-based algorithm is mainly determined by the amount of input data[1][2], i.e., the number of edge pixels, and skeletonization shrinks the detected edges to unit-width edges. This minimizes the amount of input data[1].

Usually, the skeletonization employs the morphological methods which are drastically subject to the size of structure element(also called mask)[3], thus the results may not well retain the general shapes of objects. An alternative solution has been provided by Davies[1]. The distance function is introduced to guide the thinning algorithm which generates unit-width homotopic skeletons[2], but there are isolated points scattering throughout the resulting image when the input image is taken from the real world. Such points provide poor information about the general shape and even add some difficulty to the following analysis. Therefore, the isolated points should be removed and this is achieved by introducing an additional condition of thinning in the algorithm.

To merge with the technical trend, the algorithm is implemented on Windows platforms by using IDE(Integrated Development Environment) named VS2008(Visual Studio 2008). Among all programming languages supported by VS2008, C# is chosen as the actual programming language on account of its powerful features provided directly by Windows producer Microsoft.

During the implementation, several problems arises one after another. Some difficult problems are how to implement logical operations of numbers belonging to the value type Double and how to save computations in subroutines of the thinning algorithm. The former is solved by employing extension methods which are features of C# and the latter is handled by interchanging pointers of image matrices. These may be the best solutions in this particular situation. Finally, all routines are integrated into the class ImageProcessing.

With the help of class ImageProcessing, an application software is successfully developed and the data of skeletonized binary image is obtained and visualized.

2. METHODS

The guided thinning algorithm mainly consists of two parts. The first is finding the set of local maxima of distance and the second is thinning objects based on the set. Finding local maxima involves manipulation of the distance function, and thinning is actually composed by stripping points in four directions in the image space and finally removing spurs generated by the stripping.

The whole procedure is shown in the following activity diagram of Unified Modeling Language(UML)[4][5].

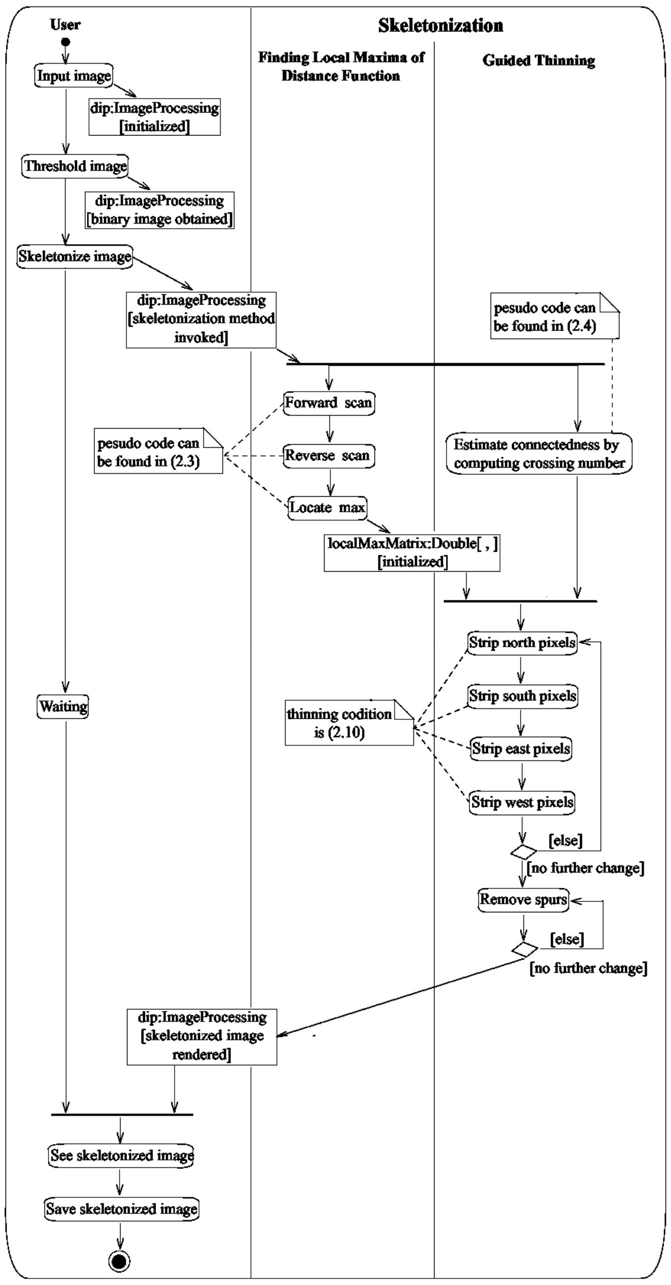

User
Skeletonization
Finding Local Maxima of Distance Function
Guided Thinning
Input image
dip:ImageProcessing [initialized]
Threshold image
dip:ImageProcessing [binary image obtained]
Skeletonize image
dip:ImageProcessing [skeletonization method invoked]
pesudo code can be found in (2.4)
Forward scan
Reverse scan
Locate max
pesudo code can be found in (2.3)
Estimate connectedness by computing crossing number
localMaxMatrix:Double[,] [initialized]
Strip north pixels
Strip south pixels
Strip east pixels
Strip west pixels
thinning codition is (2.10)
Waiting
[else]
[no further change]
Remove spurs
[else]
[no further change]
dip:ImageProcessing [skeletonized image rendered]
See skeletonized image
Save skeletonized image

FIGURE 1: The activity diagram of skeletonization..

2.1 Finding Local Maxima

The concept of distance function is quite simple, it just requires every pixel is labeled by an integer representing the distance from the background. This can be summarized as following[2]:

$$\forall p \in X, \quad \mathrm{dist}(p) = \min\{\, n \in N, p \text{ not in } (X \bullet nB) \,\} \tag{2.1}$$

For each pixel *p* in the object *X*, its distance is given by *dist(p)*. Operator $\bullet$, notations *B* and *N* respectively denote the erosion operation, the object and an integer which represents the maximal number of the erosion operations shrinking *B* to single points. Therefore, *dist(p)* assigns the integer *i* to *p* when *p* is removed by the *i*th erosion.

The distance is measured by using two raster scans, the forward and reverse scans. For making the explanation more comprehensible, every pixel in a 3-by-3 mask is denoted by one of notations *A0, A1, ... , A8* as following:

$$\begin{bmatrix} A4 & A3 & A2 \\ A5 & A0 & A1 \\ A6 & A7 & A8 \end{bmatrix} \tag{2.2}$$

For the forward scan, the mask regularly moves from left to right and top to bottom in an image and stops when the central notation A0 is of non-zero value to find the minimum of A2, A3, A4 and A5, then changes the value of central pixel to the sum of the minimum and 1 and move to next pixel. The scan is sequential which implies the current value-modified pixel is involved in the sequential value modification of the adjacent pixel next to the current one. Hence, there is only one image space for simultaneously reading data to process and storing the resultant data, unlike the usual parallel algorithm which requires two independent congruent image spaces to separate the two procedures for keeping the input data unchanged during processing. After the scan, because the procedure just checks A2, A3, A4 and A5, the integers as distance markers increase gradually from left to right and up to down in the inner area of an object.

If the left image of the following figure is the input binary image, then the right image is the visualized processed data.

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 2 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 1 1    0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 2 1 1
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0    0 0 0 0 0 0 0 0 0 0 0 0 1 1 2 2 2 1 1 1 0 1 1 2 2 2 0
0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1    0 0 0 0 0 0 0 0 0 0 0 1 1 2 2 3 2 2 2 1 1 1 2 2 3 1 1
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0    0 0 0 0 0 0 0 0 0 0 1 1 2 2 3 3 3 3 2 2 2 2 2 3 2 2 0
0 0 0 0 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0    0 0 0 0 0 1 0 0 1 1 1 2 2 3 3 4 4 3 3 3 3 3 3 3 3 0 0
0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0    0 0 0 0 1 1 1 1 1 2 2 2 3 3 4 4 4 4 4 4 4 4 4 4 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0    0 0 0 1 1 2 2 2 2 2 3 3 3 4 4 5 5 5 5 5 5 5 5 0 0 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0    0 0 1 1 2 2 3 3 3 3 3 4 4 4 5 5 6 6 6 6 6 6 0 0 0 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0    0 0 1 2 2 3 3 4 4 4 4 4 5 5 5 6 6 7 7 7 7 0 0 0 0 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0    0 0 1 2 3 3 4 4 5 5 5 5 5 6 6 6 7 7 8 8 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0    0 0 1 2 3 4 4 5 5 6 6 6 6 6 7 7 7 8 8 1 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0    0 0 1 2 3 4 5 5 6 6 7 7 7 7 7 8 8 8 2 1 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0    0 0 1 2 3 4 5 6 6 7 7 8 8 8 8 8 9 3 2 0 0 0 0 0 0 0 0
0 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0    0 1 1 2 3 4 0 0 0 0 1 2 3 4 5 6 4 3 0 0 0 0 0 0 0 0 0
0 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0    0 1 2 2 0 0 0 0 0 0 0 0 1 2 3 4 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0    1 1 2 1 0 0 0 0 0 0 0 0 1 2 3 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0    1 2 2 0 0 0 0 0 0 0 0 1 1 2 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0    0 1 0 0 0 0 0 0 0 0 0 1 2 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0 0 1 1 1 2 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0 0 1 2 2 2 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

FIGURE 2: Thinning process 1.

Obviously, pixels in the lower right area of the object are labeled by wrong integers. This can be fixed by applying a reverse scan to it. Reverse raster scan works exactly as the forward scan except it moves the mask from right to left and bottom to top, i.e., in an inverse direction with respect to the forward scan. The result is visualized as the left part of the following figure:

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 2 0    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 2 1 1    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 2 2 2 1 1 1 0 1 1 2 2 1 0    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 0 0
0 0 0 0 0 0 0 0 0 0 0 1 1 2 2 3 2 2 2 1 1 1 2 2 2 1 1    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 2 2 2 0 0
0 0 0 0 0 0 0 0 0 0 1 1 2 2 3 3 3 3 2 2 2 2 2 2 1 1 0    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 0 0 0
0 0 0 0 0 1 0 0 1 1 1 2 2 3 3 4 4 3 3 3 3 2 2 1 1 0 0    0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 4 4 0 0 3 3 0 2 0 0 0 0
0 0 0 0 1 1 1 1 1 2 2 2 3 3 4 4 4 4 3 3 2 2 1 1 0 0 0    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 3 0 0 0 0 0 0 0
0 0 0 1 1 2 2 2 2 2 3 3 3 4 4 5 4 3 3 2 2 1 1 0 0 0 0    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 2 2 3 3 3 3 3 4 4 4 5 5 4 3 2 2 1 1 0 0 0 0 0    0 0 0 0 0 0 0 0 0 0 0 0 0 0 5 5 0 0 0 0 0 0 0 0 0 0 0
0 0 1 2 2 3 3 4 4 4 4 4 5 5 5 4 4 3 2 1 1 0 0 0 0 0 0    0 0 0 0 0 0 0 4 4 4 4 0 5 5 5 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 2 3 3 4 4 4 4 4 4 4 4 4 4 3 3 2 1 0 0 0 0 0 0 0    0 0 0 0 3 0 4 4 4 4 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 2 3 3 3 3 3 3 3 3 3 4 4 3 3 2 2 1 0 0 0 0 0 0 0    0 0 0 0 3 0 0 0 0 0 0 0 0 4 4 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 2 2 2 2 2 2 2 2 2 3 3 3 3 2 2 1 1 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 2 2 1 1 1 1 1 1 2 2 2 2 2 2 1 1 0 0 0 0 0 0 0 0    0 0 0 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 0 0 0 0 1 1 1 2 2 1 1 1 0 0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 0 0 0 0 0 0 0 0 0 0 0 0
0 1 2 1 0 0 0 0 0 0 0 0 1 2 1 1 0 0 0 0 0 0 0 0 0 0 0    0 0 2 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0    0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 1 2 1 0 0 0 0 0 0 0 0 0 0 0 0 0    0 1 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 1 1 2 1 0 0 0 0 0 0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0 0 1 1 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

FIGURE 3: Thinning process 2.

After the distances of every pixel are measured by these two scans, finding local maxima can eventually be performed by using a parallel subroutine. It checks the neighborhood of the central pixel in a 3-by-3 mask in one image space, finds the maximum and compares it with the value of the center. If the value is larger than the maximum, then the value is recorded at the corresponding coordinate of a separate congruent space, otherwise 0 is recorded. This generates a result shown in the right part of fig.3.

The general procedure of two raster scans and finding maxima can be summarized as the following pseudo code:

$$\begin{aligned}
&\text{forward scan : if}(A0 > 0) \quad A0 = \min(A2, A3, A4, A5) + 1;\\
&\text{reverse scan : if}(A0 > 0) \quad A0 = \min(A6, A7, A8, A1) + 1;\\
&\text{locate max : if}(A0 > 0 \,\&\&\, A0 > \max(A1, A2, A3, A4, A5, A6, A7, A8)\\
&\qquad\qquad B0 = A0; \quad \text{else} \quad B0 = 0;
\end{aligned} \tag{2.3}$$

Note the direction of moving mask in curly brackets of reverse scan is the inverse direction of forward scan and locate max, and the raster scans are sequential and locate max is parallel.

2.2 Guided Thinning

2.2.1. Crossing Number

Once the maxima are found and recorded, it's possible to design and implement guided thinning algorithm. The size of the mask is assumed to be 3-by-3 as (2.2) in finding maxima. In such a small mask, the crossing number χ (chi)[1] is employed to judge connectedness of A0.

The χ is obtained by summing the times of value changing during the travel which starts from an arbitrary pixel in the neighborhood of A0 clockwise or anticlockwise and ends when the same pixel is visited again. Under this definition, the integer χ can't be odd. To correctly compute χ, three logical operators are necessary, i.e., &&(AND) operator, !=(NOT EQUAL) operator and !(NOT) operator, and the connectedness criterion must be 8-connectedness.

Firstly, the value changing is checked along four diagonal directions, namely, A1 to A3, A3 to A5, A5 to A7 and A7 to A1. For each diagonal, if two values of Ai's are different, χ is then added by 1 to denote one time of value changing.

Secondly, four corners are checked, e.g., A3, A4 and A5 form the upper left corner. For each corner, if the case is that the corner pixel is 1 and the other two are 0, then χ is added by 2. This case can't be detected by checking diagonals. These two checks complement each other and there's no intersection between them. In pseudo code, this can be expressed as:

$$
\begin{aligned}
&chi = (A1 \mathrel{!}= A3) + (A3 \mathrel{!}= A5) + (A5 \mathrel{!}= A7) + (A7 \mathrel{!}= A1) + \\
&2 * \{(!A1 \,\&\&\, A2 \,\&\&\, !A3) + (!A3 \,\&\&\, A4 \,\&\&\, A5) + \\
&(!A5 \,\&\&\, A6 \,\&\&\, !A7) + (!A7 \,\&\&\, A8 \,\&\&\, !A1)\}.
\end{aligned}
\tag{2.4}
$$

Clearly, the larger the χ is, the more intense the connectedness of the central pixel is. The marginal value of χ separates the cases the central pixel is removed or preserved is 2 inasmuch as such pixels are located on the edge of an object and appropriate to be removed. There are two cases causing ambiguity when χ equals 2, they are exemplified and shown in the following figure.

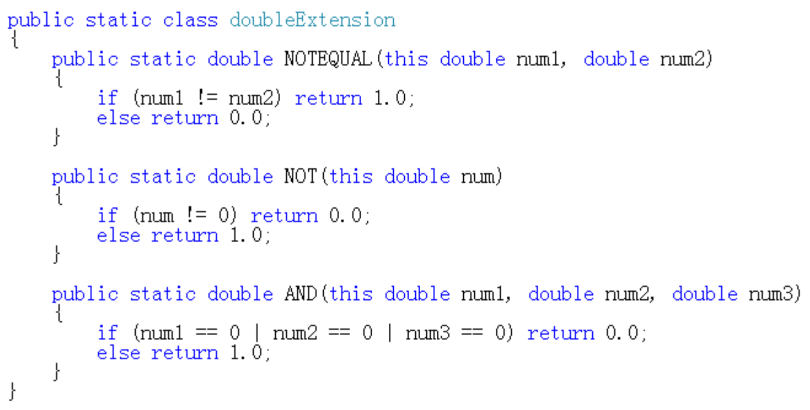
$$
\begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix} \qquad\qquad \begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{matrix}
\tag{2.5}
$$

The left matrix of (2.5) represents an end point of an unit-width line which possibly denotes a part of the final skeleton, so the central point in this case can’t be removed. Therefore, an additional condition of thinning has to be made, i.e., the sum of values of neighborhood doesn’t equal 1. The conditions of a common thinning are summarized as following:

$$
(chi == 2) \,\&\&\, (sumOfNeighborhood \mathrel{!}= 1)
\tag{2.6}
$$

2.2.2. Extension Methods for Computing Crossing Number

Only the logical operator &&(AND) shown in (2.4) is valid for Byte values and the !=(NOT EQUAL) operator and !(NOT) operator both return Boolean values. This complicates programming, especially when the computation of converting data type of matrix is expected to be saved. One of the possible solutions is using extension methods[6] which extend the existing class by adding new methods. In this paper, all matrices involved in calculation are assumed to be of Double[,] for programming convenience and saving computation of converting data types. The extension methods of Double are organized as a static class to meet the requirement of C#, and the actual code is shown in the following figure:



```
public static class doubleExtension
{
    public static double NOTEQUAL(this double num1, double num2)
    {
        if (num1 != num2) return 1.0;
        else return 0.0;
    }

    public static double NOT(this double num)
    {
        if (num != 0) return 0.0;
        else return 1.0;
    }

    public static double AND(this double num1, double num2, double num3)
    {
        if (num1 == 0 | num2 == 0 | num3 == 0) return 0.0;
        else return 1.0;
    }
}
```

FIGURE 4: Code of Extension methods.

The code of χ is then simplified as shown in the following:

```
double[,]thinningMatrix = (double[,])sourceMatrix.Clone();
int height = matrix.GetLength(0);
int width = matrix.GetLength(1);
int upperLeft = 0;
int upperMiddle = 1;
int upperRight = 2;
int left = width;
int center = width + 1;
int right = width + 2;
int lowerLeft = 2 * width;
int lowerMiddle = 2 * width + 1;
int lowerRight = 2 * width + 2;
fixed (double* pointer3 = thinningMatrix){
chi = pointer3[left].NOTEQUAL(pointer3[upperMiddle]) +
      pointer3[left].NOTEQUAL(pointer3[lowerMiddle]) +
      pointer3[right].NOTEQUAL(pointer3[upperMiddle]) +
      pointer3[right].NOTEQUAL(pointer3[lowerMiddle]) +
      2 * (pointer3[upperRight].AND(pointer3[right].NOT(), pointer3[upperMiddle].NOT()) +
           pointer3[upperLeft].AND(pointer3[upperMiddle].NOT(), pointer3[left].NOT()) +
           pointer3[lowerLeft].AND(pointer3[left].NOT(), pointer3[lowerMiddle].NOT()) +
           pointer3[lowerRight].AND(pointer3[lowerMiddle].NOT(), pointer3[right].NOT()));
}
```

FIGURE 5: Code of χ.

As shown in fig.5, extension methods AND(), NOT() and NOTEQUAL() are envisaged as the methods of Double which can be directly invoked by instances of Double, i.e., Double values.

2.2.3. Thinning Based on Local Maxima

The clusters of maxima shown in the left part of fig.3 consist of pixels in the central area of an object. These pixels are of very high possibility to be parts of the skeleton and they represent the general shape of the object in some degree. Hence, these pixels shouldn't be removed during the thinning procedure.

As the raster scans, thinning is sequential. The sequentiality introduces a new problem that is the final skeleton would be biased towards the bottom of the image inasmuch as the direction of regular moving the mask causes pixels in the upper left area of an object are continuously removed until the mask reaches the lower right edge of the object. Thus, the removal of pixels shouldn't be executed once, it should be executed in several times. For each time, the pixels in the circumambient ring of the area of the object are removed if the condition is satisfied. This can be summarized as the following formula[2]:

$$X \otimes \{B_{(i)}\} = (((X \otimes B_{(1)}) \otimes B_{(2)})... \otimes B_{(n)}) \tag{2.7}$$

The notation $\{B_{(i)}\}$ denotes the Golay alphabet[7] and operator denotes the stripping operation. Here n is 4, and $B_{(1)}$, $B_{(2)}$, $B_{(3)}$, $B_{(4)}$ are shown in the following figure:

$$\begin{bmatrix} * & 0 & * \\ * & 1 & * \\ * & 1 & * \end{bmatrix} \begin{bmatrix} * & 1 & * \\ * & 1 & * \\ * & 0 & * \end{bmatrix} \begin{bmatrix} * & * & * \\ 0 & 1 & 1 \\ * & * & * \end{bmatrix} \begin{bmatrix} * & * & * \\ 1 & 1 & 0 \\ * & * & * \end{bmatrix} \tag{2.8}$$

Notation * denotes the pixel whose value can be 0 or 1. Actually, $B_{(1)}$ represents north pixels. The corresponding pseudo code is as following.

```
do{strip north pixels;
    strip south pixels;
    strip east pixels;                (2.9)
    strip west pixels}
until no further change
```

Combining this detection of directional pixels and (2.6), the condition for stripping directional pixels is obtained. That is:

$$\begin{aligned}&(A0>0)\ \&\&\ (A0 \notin local\ \max ima)\ \&\&\ (A0 \in directional\ \ pixels)\\&\&\&\ (chi == 2)\ \&\&\ (sum\ of\ \ neighborhood\ != 1)\end{aligned} \qquad (2.10)$$

The result of applying (2.9) and (2.10) is shown in the left part of the fig.6.

Obviously, the object is not completely skeletonized, e.g., the upper right corner of the skeleton is 2-pixel wide. This is because these non-unit wide parts are maxima shown in the right part of the fig.3 which can't be removed according to the condition (2.10). In addition, the algorithm will generate unexpected spurs. Thus, a final sequential thinning with a modified condition (2.10) is necessary. The condition is:

$$(A0>0)\ \&\&\ (chi == 2)\ \&\&\ (sum\ of\ \ neighborho\ od\ != 1)\,. \qquad (2.11)$$

The final result is shown in the right part of the following figure.

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 1 1 1 1 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 0 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

FIGURE 6: Thinning process 3.

This result is quite successful. The general shape of the object is preserved and there are no broken lines based on 8-connectedness criterion.

2.2.4. Amelioration of The Guided Thinning Algorithm

A detail of (2.9) should be noticed, that is how the four stripping subroutines are combined. For a stripping subroutine, there should be at least two image spaces to store the input data generated by the previous subroutine and the output data processed by the current subroutine. A possible solution may be that keeping one space constantly store the input data and the other only store the output data. This is problematic on account of that another image space has to be introduced to complete the interchange of the two spaces.

A much better solution is just interchanging the pointers of two spaces before a stripping subroutine is triggered. Assuming a pointer ptr1 points to a copy of the original binary image and another pointer ptr2 points to a congruent image space, the matrix with ptr1 is fed into the subroutine "strip north pixels" and the processed data is stored in the matrix with ptr2. For the next subroutine "strip south pixels", the input data should be the matrix with ptr2 and the processed data can be stored in the matrix with ptr1. For now, the data of original binary image storing in the matrix with ptr1 is useless for the further processing in (2.9). There are four such interchanges in one loop.

Although the interchange can't change the nature of (2.9), i.e., it roughly is O(n2) where n denotes one dimension of the image space, it saves 8*n2 times of value assignments and a storage of an image space.

For the ideal binary image as fig.2, this result is satisfactory. But for real images, isolated points scatter throughout the whole image. These points provide little information about the skeleton and may make the following analysis difficult. Thus, they should be eliminated. This is achieved by modifying the condition (2.11) of the final spur-removing subroutine. The condition is:

$$(A0>0)\,\&\&\,(chi==2\,||\,chi==0)\,\&\&\,(sum\ of\ \ neighborho\ od\ !=1)\,. \qquad (2.12)$$

3. RESULTS

The application software[8] was executed in Windows XP Professional SP3 on a laptop with Intel Core(TM)2 Duo 1.6 GHz CPU. The left of the following figure is a screenshot of the application when a color image whose size is 800×600 had been loaded and the right is the visualized result of applying the Sobel operator.

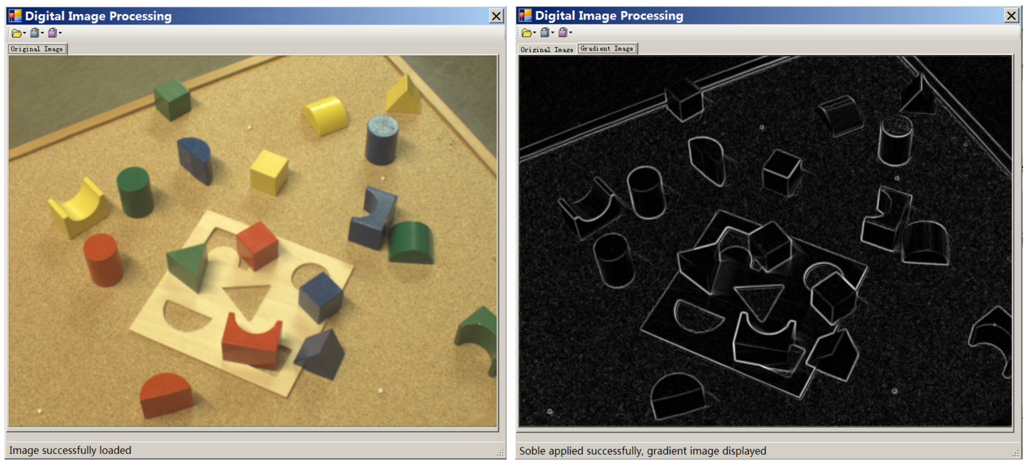
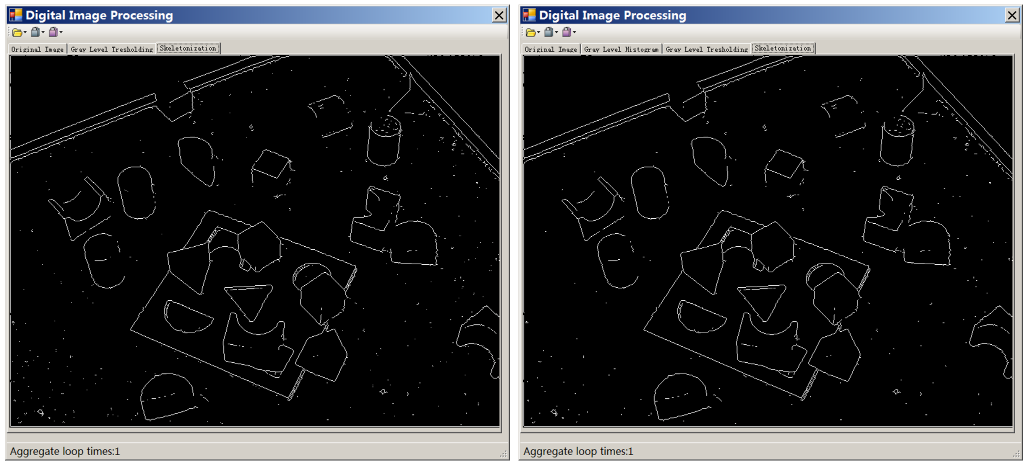


Image successfully loaded

Soble applied successfully, gradient image displayed

FIGURE 7: A loaded image and its gradient image.

The left of the following figure is the skeletonized binary image generated by the algorithm with (2.11) and the right is generated by the algorithm with (2.12).

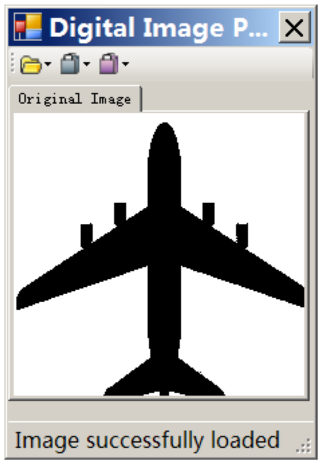


Aggregate loop times:1

Aggregate loop times:1

FIGURE 8: Binary image and its skeletonized version.

Obviously, white points scattering in the left lower corner of the left figure of fig.9 vanish in the right figure of fig.9. Another example is shown in the following figures.



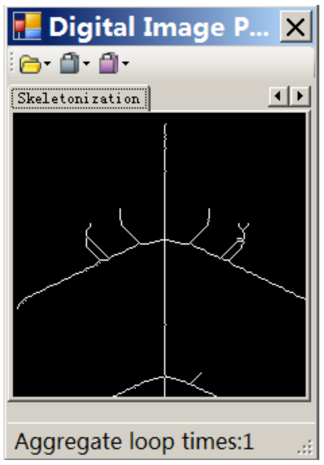


FIGURE 9: The silhouette of a plane and its skeletonized version.

4. CONCLUSION

For practical purpose, the ameliorated guided thinning algorithm is implemented on Windows platforms by using IDE named VS2008. Among all programming languages supported by VS2008, C# is chosen as the actual programming language on account of its powerful features provided directly by Windows producer Microsoft. Since the VS2008 lacks means of performing pixel-level operations, a class named ImageProcessing is developed, and the guided thinning algorithm is encapsulated as a method by the class. The problems arising in the development are overcome by employing extension methods, interchange of pointers and modifying the thinning condition. The programming procedure is thus simplified, and both the computation and storage are saved to the full extent, but the nature of the algorithm is not deeply improved. Hence the performance and spur removal can be further ameliorated. This may require a larger mask and subtle design of the thinning conditions.

5. REFERENCES

[1] E. R. Davies, Machine Vision: Theory, Algorithms, Practicalities 3rd ed. San Fransisco, CA: Morgan Kaufmann, 2005.

[2] M. Sonka, V. Hlavac, R. Boyle, Image Processing, Analysis, and Machine Vision 3rd ed. CT: 2008.

[3] R. C. Gonzalez, R. E. Woods, Digital Image Processing 3rd ed. Upper Saddle River, NJ: Prentice Hall, 2007.

[4] I. Jacobson, G. Booch, and J. Rumbaugh, Unified Modeling Language User Guide 2nd ed. Boston, MA: Addison-Wesley, 2005.

[5] J. Rumbaugh, I. Jacobson, and G. Booch, Unified Modeling Language Reference Manual 2nd ed. Boston, MA: Addison-Wesley, 2010.

[6] J. Albahari, B. Albahari, C# 4.0 In A Nutshell 4th ed. Sebastopol, CA: O'Reilly, 2010.

[7] J. Serra, Image Analysis and Mathematical Morphology, London: Academic Press, 1982.

[8] L. Jia, Y. Sun, M. Wang, Y. Gu, “A Research on Implementation of Image Scattergram by Using C#”, 2011 International Conference on System Design and Data Processing(ICSDDP 2011), IEEE press, February. 2011, pp. 353- 355.