IOSR Journal of Computer Engineering (IOSRJCE)
ISSN: 2278-0661 Volume 2, Issue 5 (July-Aug. 2012), PP 38-42
www.iosrjournals.org

A Novel Approach To Topological Skeletonization Of English Alphabets And Characters

Chinmay Chinara[1], Nishant Nath[2], Subhajeet Mishra[3]
[1, 2, 3]*(Dept. of ECE, S'O'A University, India)*

Abstract : *In this paper we put forward a modified approach towards skeletonization of English alphabets and characters. This algorithm has been designed to find the skeleton of all the typeface of Modern English as present in the Microsoft database. The algorithm has been kept simple and optimized for efficient skeletonization. Finally, the performance of the algorithm after testing has been aptly demonstrated.*

Keywords *–Digital Library*, *Microsoft Visual C++, Skeletonization, Structuring Element, Thresholding, Typeface*

I. INTRODUCTION

Character recognisation also known as *Optical Character Recognisation (OCR)* is one of the major subset of pattern recognisation. Over the years, offline character recognisation has been achieving great demand due to evolution in the field of *digital library* and *banking.* Skeletonization and thinning are amongst the major aspects of pre-processing steps for the correct working of OCR systems. Skeletonization is a process for reducing foreground regions in a binary image to a skeletal remnant that largely preserves the extent and connectivity of the original region while throwing away most of the original foreground pixels [8][5]. In other words, it is like the loci of intersecting waves emanating from different points of a branch. Thinning is done to reduce data storage by converting the binary image into a skeleton or a line drawing [5]. The main objective; however is to retain the topographical properties of the alphabets and characters.

The complexity in the skeletonization of Modern English Alphabets and Characters lies in its vastness of typeface. This paper proposes an effective skeletonization algorithm that is compatible with the entire typeface present in the Microsoft database.

II. BACKGROUND – MODERN ENGLISH

The modern English alphabets are a sub-group of the 26 Latin alphabets set by the *International Organization of Standardization (ISO).* The shape of these alphabets depends on the *typeface,* commonly known as the font of the alphabet. The above 26 alphabets are represented in two forms, one being the *Majuscule* form (also called *Uppercase* or *Capital* letters) and the other being the *Minuscule* form (also called *Lowercase* or *Small* letters). These are again subdivided into two parts, namely *Vowels* and *Consonants.* The characters consist of a vast database and they vary from typeface to typeface. The few alphabets and characters over which skeletonisation was implemented are shown in *Fig-(1-3).*They are based on the *Arial* font of the size *22.*

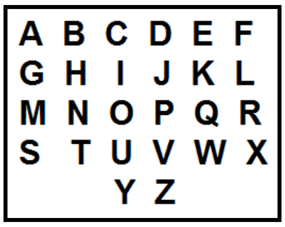


Fig-1. Majuscule form of alphabets

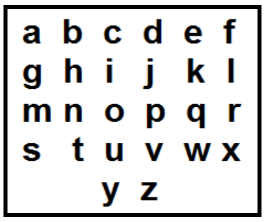


Fig-2. Minuscule form of alphabets

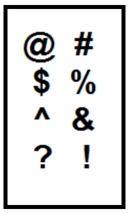


Fig-3. Few special characters

III. LITERATURE SURVEY

Many skeletonization algorithms are proposed in the literature. Till date, all the skeletonization algorithms work very well with English alphabets and letters. A properly skeletonised and thinned image aids in segmentation and feature extraction, which are crucial factors for character recognition. The skeletonization algorithms are mainly classified in two groups: one on the basis of distance transforms and other on the basis of thinning approaches.

The paper by G. Sanniti di. Baja [2] used distance transform for the purpose of skeletonization. Multiple erosions with suitable structuring element were implemented here until all the foreground regions of the image had been eroded away.

The paper by Gisela Klette[3] showed a detailed study of the various distance transform techniques and thinning approaches used to attain skeletonization. This study indicated that *Iterative thinning algorithms* are the most efficient ones.

The paper by Aarti Desai et al. [1] reflects a kind of thinning algorithm in which comparison with different structuring elements and elimination of unnecessary black pixels is the key. This, when modified gives a very good skeletonized output, thus making it the gateway to our proposed algorithm.

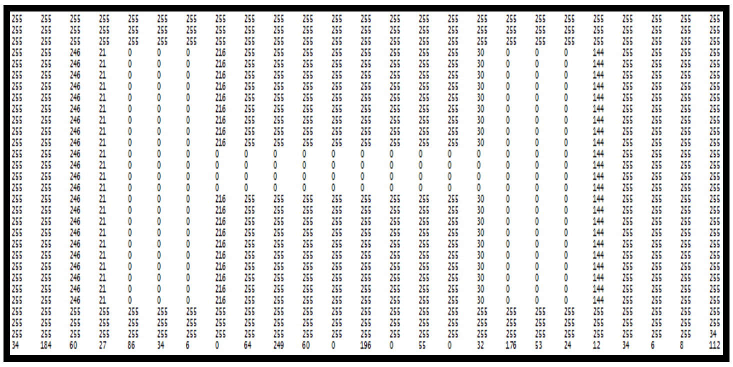
Paper by [4] showed an efficient use of iterative algorithms. Comprehensive work by [6] and [7] also helped us understand the morphological operations in greater details.

IV. PROPOSED ALGORITHM

The proposed algorithm is a modification of the work done on thinning by Aarti Desai et al. [1] and is implemented for the Topological Skeletonization of English Alphabets and Characters. This involves four stages: binarization, adding of dummy white pixels, creating structuring elements & applying it to the image and removing the noise to obtain the final skeletonized image.

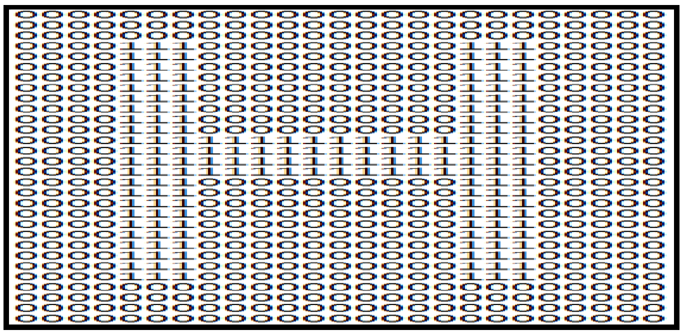
4.1 Binarization

This process involves the conversion of the input bitmap image to a black and white image by setting a particular threshold level. The image is then converted to binary by representing the background (white pixel) by ‘0’ and foreground (black pixel) by ‘1’. This is shown in *Fig-4.* and *Fig-5.*



255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	246	21	0	0	0	216	255	255	255	255	255	255	255	255	30	0	0	0	144	255	255	255	255
255	255	246	21	0	0	0	216	255	255	255	255	255	255	255	255	30	0	0	0	144	255	255	255	255
255	255	246	21	0	0	0	216	255	255	255	255	255	255	255	255	30	0	0	0	144	255	255	255	255
255	255	246	21	0	0	0	216	255	255	255	255	255	255	255	255	30	0	0	0	144	255	255	255	255
255	255	246	21	0	0	0	216	255	255	255	255	255	255	255	255	30	0	0	0	144	255	255	255	255
255	255	246	21	0	0	0	216	255	255	255	255	255	255	255	255	30	0	0	0	144	255	255	255	255
255	255	246	21	0	0	0	216	255	255	255	255	255	255	255	255	30	0	0	0	144	255	255	255	255
255	255	246	21	0	0	0	216	255	255	255	255	255	255	255	255	30	0	0	0	144	255	255	255	255
255	255	246	21	0	0	0	216	255	255	255	255	255	255	255	255	30	0	0	0	144	255	255	255	255
255	255	246	21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	144	255	255	255	255
255	255	246	21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	144	255	255	255	255
255	255	246	21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	144	255	255	255	255
255	255	246	21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	144	255	255	255	255
255	255	246	21	0	0	0	216	255	255	255	255	255	255	255	255	30	0	0	0	144	255	255	255	255
255	255	246	21	0	0	0	216	255	255	255	255	255	255	255	255	30	0	0	0	144	255	255	255	255
255	255	246	21	0	0	0	216	255	255	255	255	255	255	255	255	30	0	0	0	144	255	255	255	255
255	255	246	21	0	0	0	216	255	255	255	255	255	255	255	255	30	0	0	0	144	255	255	255	255
255	255	246	21	0	0	0	216	255	255	255	255	255	255	255	255	30	0	0	0	144	255	255	255	255
255	255	246	21	0	0	0	216	255	255	255	255	255	255	255	255	30	0	0	0	144	255	255	255	255
255	255	246	21	0	0	0	216	255	255	255	255	255	255	255	255	30	0	0	0	144	255	255	255	255
255	255	246	21	0	0	0	216	255	255	255	255	255	255	255	255	30	0	0	0	144	255	255	255	255
255	255	246	21	0	0	0	216	255	255	255	255	255	255	255	255	30	0	0	0	144	255	255	255	255
255	255	246	21	0	0	0	216	255	255	255	255	255	255	255	255	30	0	0	0	144	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	34
34	184	60	27	86	34	6	0	64	249	60	0	196	0	55	0	32	176	53	24	12	34	6	8	112

Fig-4. The letter ‘H’ before thresholding and binarisation



0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	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	1	1	1	0	0	0	0	0
0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0
0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0
0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0
0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0
0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0
0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0
0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0
0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	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	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	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	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	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0
0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0
0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0
0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0
0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0
0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0
0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0
0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0
0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0
0	0	0	0	1	1	1	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	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fig-5. The letter ‘H’ after thresholding and binarisation

4.2 Addition of Dummy Pixels

When the structuring element is applied to the binary image directly, then for the pixels present in the extreme left, right, top and bottom sides of the image, we get an erroneous, non-preexisting pixel values. To eliminate this, we add an extra layer of white pixels (dummy layer) to each of the above mentioned sides as shown in *Fig-6.* No operation is done over these pixels; rather they just provide assistance to the process of comparison with the structuring elements. This binary image is the base image that is to be skeletonized.

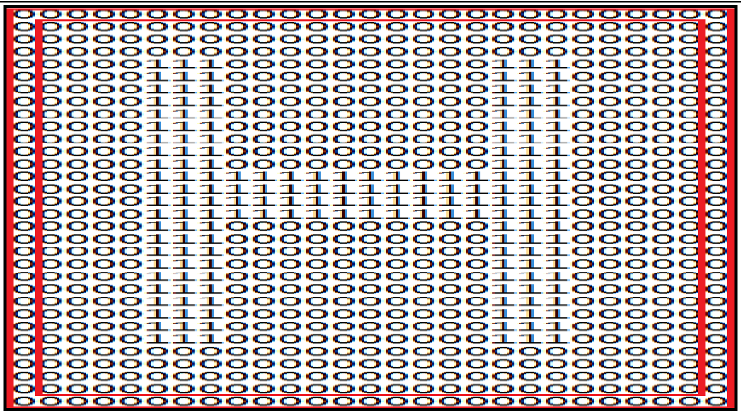


Fig-6. The letter 'H' after adding dummy pixels (given by area under the red concentric region)

4.3 Creation of structuring elements and application to the image

Eight 3X3 structuring elements of the form shown in *Fig-(7-14).* are applied to the base image. The application involves traversing the base image, pixel by pixel where *(m, n)* represents the current position of the pixel under consideration. '*m'* & *'n'* represent the row and the column positions of the pixel under consideration from the base image. The comparison is done only if the *(m, n)* pixel is black. The *(m, n)* white pixel is left unchanged.

0	0	0
X	①	X
1	1	1

Fig-7. 1st S.E.

X	0	0
1	①	0
1	1	X

Fig-8. 2nd S.E.

1	X	0
1	①	0
1	X	0

Fig-9. 3rd S.E.

1	1	X
1	①	0
X	0	0

Fig-10. 4th S.E.

0	0	X
0	①	1
X	1	1

Fig-11. 5th S.E.

1	1	1
X	①	X
0	0	0

Fig-12. 6th S.E.

X	1	1
0	①	1
0	0	X

Fig-13. 7th S.E.

0	X	1
0	①	1
0	X	1

Fig-14. 8th S.E.

The *'0'* in the above figures represent the background (white) and the *'1'* represents the foreground (black). The *'X'* pixel is not considered for comparison. The encircled region represents the central element that has to be placed over the *(m, n)* black pixel of the base image for comparison. The comparison is done between the adjacent pixels of the pixel under consideration of the base image and the adjacent pixels of the central pixel of the structuring element. If each of the respective pixels match then the *(m, n)* pixel of the base image is converted to white. This comparison and replacement procedure is repeated for each and every structuring element with the base image till no further changes are possible. The skeletonized image obtained however contains noise as shown in *Fig-15.* which has to be removed to get the desired skeleton.

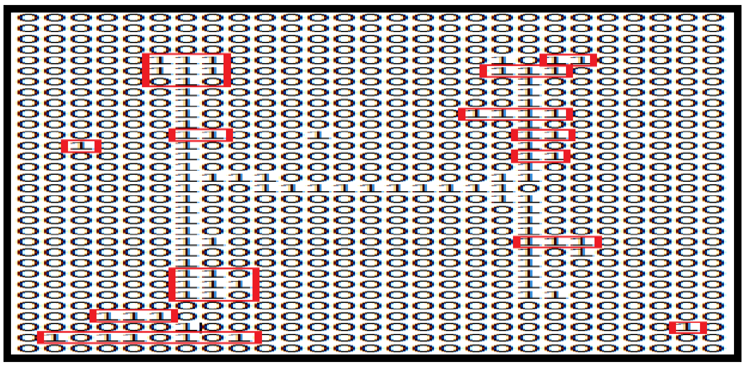


Fig-15. The skeletonized letter 'H' after comparing with structuring elements (contains noise given by red-boxed regions)

4.4 Removal of noise

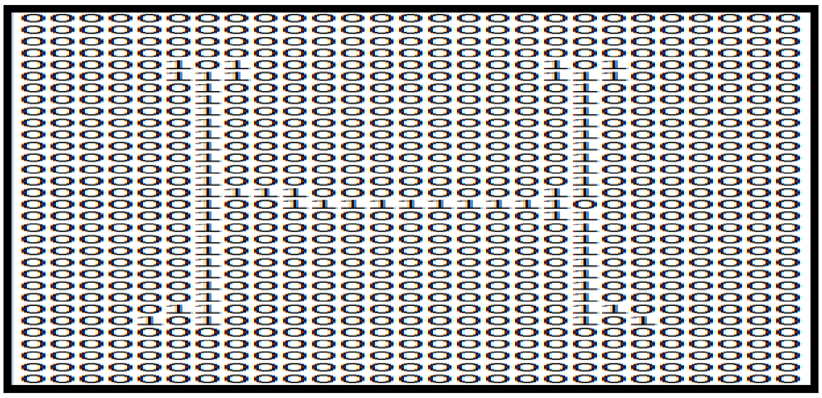
(m-1), (n-1)	**(m-1), n**	**(m-1), (n+1)**
m, (n-1)	**m, n**	**m, (n+1)**
(m+1), (n-1)	**(m+1), n**	**(m+1), (n+1)**

Fig-16. The 3X3 window frame for the considered (m, n) pixel and its surrounding eight pixels

Removal of noise solely depends on whether a black pixel can 'safely' be converted into a white pixel. The white pixels are kept unchanged as the noise occurs due to presence of extra black pixels only. For achieving this we browse the surrounding of the black pixel at *(m, n)* position i.e. we count the number of black pixels and white pixels around *(m, n)* black pixel. If the number of black pixels is less than or equal to 2 then the *(m, n)* black pixel is a safe one and is left unchanged. If the number is greater than 2 then we count the number of *[white (minus) black]* colour combinations around the *(m, n)* black pixel considered. If the number is not equal to 1 then we leave the *(m, n)* black pixel unchanged but if the number is equal to 1 then that particular *(m, n)* black pixel is converted to white if and only if the following conditions satisfy:

i. (m, n+1) or (m+1, n) or both (m-1, n) and (m, n-1) are white.
ii. (m-1, n) or (m, n-1) or both (m, n+1) and (m+1, n) are white.

After applying the above noise removal technique we get the final desired skeletonized image as shown in *Fig-17.*



```
000000000000000000000000000
000000000000000000000000000
000000000000000000000000000
000000000000000000000000000
000001010000000000101000000
000001110000000000111000000
000000100000000000010000000
000000100000000000010000000
000000100000000000010000000
000000100000000000010000000
000000100000000000010000000
000000100000000000010000000
000000100000000000010000000
000000100000000000010000000
000000100000000000010000000
000000111100000000110000000
000000100111111111100000000
000000100000000000110000000
000000100000000000010000000
000000100000000000010000000
000000100000000000010000000
000000100000000000010000000
000000100000000000010000000
000000100000000000010000000
000000100000000000010000000
000001100000000000011000000
000010100000000000010100000
000000000000000000000000000
000000000000000000000000000
000000000000000000000000000
000000000000000000000000000
000000000000000000000000000
```

Fig-17. The final skeletonized letter 'H' (after noise removal)

V. DEVELOPMENT PLATFORM

The algorithm was implemented using the *Microsoft Visual C++ (ver. 2010)* development platform. Due to its quick, enhanced and interactive environment, we were able to study, implement and analyze all the previous algorithms which resulted into our proposed algorithm.

VI. CONCLUSION

The modified algorithm applied to skeletonization of English characters works well for all type of standard fonts present in the Microsoft database. The output contains very little noise, maintains connectivity and proper branching, thus making it a good choice for the English OCR systems. The time complexity of this algorithm is also very negligible as shown in *Table-1.* and is an improvement over the algorithm suggested by [1]. The algorithm was tested on a system with *Intel Dual Core Processor, 2.0 GHz, 4GB RAM, 1GB Graphics Driver.* This algorithm can also be implemented on other languages like the *Latin, Greek, Roman* as well as Indian scripts like *Devnagari.*

Algorithm	**Time-taken (in ms)**
Aarti Desai et al. [1]	17
Proposed algorithm	15

VII. ACKNOWLEDGEMENTS

We acknowledge the research facilities and development platform provided to us at the *Video-Data Analysis Systems Lab, Electro-Optical Tracking Division, Integrated Test Range, DRDO, Chandipur, Balasore, India* for the successful completion of our research.

REFERENCES

[1] Aarti Desai, Latesh Malik and Rashmi Welekar, "A New Methodology for Devnagari Character Recognisation", JM International Journal on Information Technology, Volume-1 Issue-1, ISSN: Print 2229-6115, January 2011, pp. 56-60
[2] G. Sanniti di Baja, "Well-shaped, stable and reversible skeletons from the (3, 4)-distance transform", J. Visual Comm. Image Representation, 1994, pp. 107-115
[3] Gisela Klette, "Skeletons in Digital Image Processing", Centre for Image Technology and Robotics, Tamaki, CITR-TR-112, July 2002
[4] Khalid Saeed, Marek Tabedzki, Mariusz Rybnik and Marcin Adamski, "K3M: A Universal Algorithm for Image Skeletonization and a Review of Thinning Techniques", International Journal of Applied Mathematics and Computer Science, Volume-20, No. 2, 2010, pp. 317-335
[5] Rafael C. Gonzalez and Richard E. Woods, " Digital Image Processing", Second Edition, Prentice-Hall, 2002, pp. 541-545
[6] D. Ballard and C. Brown, "Computer Vision", Prentice-Hall, 1982, Chapter-8
[7] E. Davies, "Machine Vision: Theory, Algorithms and Practicalities", Academic Press, 1990, pp. 149 – 161
[8] R. Haralick and L. Shapiro, "Computer and Robot Vision", Vol. 1, Addison-Wesley Publishing Company, 1992, Chapter-5