IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 15, 2022, accepted April 6, 2022, date of publication April 14, 2022, date of current version April 20, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3167567

Fast Text Placement Scheme for ASCII Art Synthesis

MOONJUN CHUNG AND TAESOO KWON
Department of Computer and Software, Hanyang University, Seoul 04763, South Korea

Corresponding author: Taesoo Kwon (taesoo@hanyang.ac.kr)

This work was supported in part by the National Research Foundation of Korea (NRF) grant funded by the Korean Government [Ministry of Science and Information and Communication Technology (MSIT)] under Grant NRF-2020R1A2C1012847, and in part by the Institute of Information and Communications Technology Planning and Evaluation (IITP) grant funded by the Korean Government (MSIT) (Technology Development of Creation, Manipulation and Augmentation for XR in the Real-World Environment) under Grant 2021-0-00320.

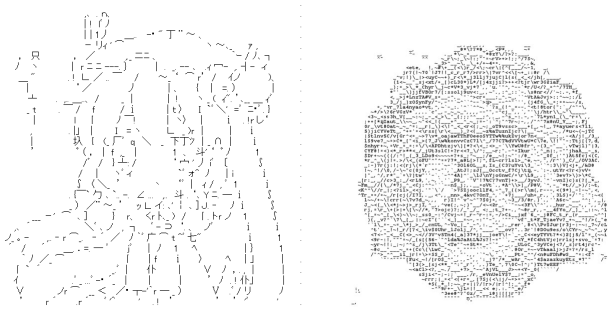
ABSTRACT This study suggests an algorithm that creates ASCII art from a binary image. Our approach aims to generate ASCII art in a short period of time using multi-threaded local optimizations for a text placement method instead of a global optimization. To generate ASCII art from various images, the original image is first converted into a thinned black and white image suitable for generating ASCII art. We then extract the pixel orientations from the input image and introduce a character similarity scheme that considers these orientations. We also propose a novel text placement algorithm to complete ASCII art in a swift manner. Our final system suggested here can generate ASCII art using a variety of proportional fonts. The results of the experiments of this study show that the suggested system can generate ASCII art much faster than existing state-of-the-art techniques using proportional fonts.

INDEX TERMS ASCII arts, black-and-white image, grayscale image, image processing.

I. INTRODUCTION

Frequent transmission of large-sized images consumes a large amount of data on networks and websites. Under a slow Internet environment, using ASCII art images can be of great use. Since ASCII art images consist of text, the data size is much smaller than that of general bitmap-based images. In addition, since ASCII art is a text-based image which also allows users to easily edit and make various revisions to it. Thanks to these characteristics, ASCII art is frequently used in web pages that do not support image uploading, and some artists create cartoon-style art by properly arranging the ASCII art images and text.

ASCII art can largely be classified into two sectors: structure-based and tone-based (see Figure 1). Structure-based ASCII art mainly represents the structure of 2D line drawing images articulated with outlines. Tone-based ASCII art, on the other hand, expresses the brightness or color of a reference image in a more realistic manner. In general, the structure-based method is a more complicated scheme than the tone-based method, as it is necessary to understand the structure of the reference image to create the ASCII art images. In this study, we propose a system for generating structure-based ASCII art.



(a) Structure-based ASCII art (b) Tone-based ASCII art

FIGURE 1. Example of structure-based ASCII art (a) and tone-based ASCII art (b). Example of tone-based ASCII art was taken from Marku *et al.* [1].

When artists manually create a structure-based ASCII art image, they usually place the appropriate characters on top of the original image. This is a time consuming and tedious task. Thankfully, there are several ways to convert reference images to structure-based ASCII art on behalf of the artists [2]–[5]. Although some of these systems show decent quality in the output, it takes a long time to run the system. Recently, an ASCII art synthesis system using a

The associate editor coordinating the review of this manuscript and approving it for publication was Lefei Zhang.

Convolutional Network (CNN) was proposed [6]. Based on machine learning, the system can reproduce the 'style' of the artist-created reference images. However, machine learning based system cannot reproduce a style that system did not learn. This means that a set of reference images are necessary for every font type or size.

In this paper, we suggest a new automatic ASCII art generation system that complements the shortcomings of other conventional methods. Our system aims to achieve the following:

- Quality: The system can create ASCII art of high quality.
- Efficiency: The results are created in a short period of time so that the artist can edit them immediately.
- Adaptability: The system can create ASCII art stably regardless of the font type or size.

Two algorithms are at the heart of our system: one is to extract the orientation features of the input image and use it for matching the most suitable characters, and another is to place such character on the input image stripes, which is called the text placement algorithm. The text placement suggested in this study was inspired by how artists manually make ASCII art. The suggested system does not just place the character images from the left to the right side of the input image stripes. Instead, it places the most suitable character in the local optimum location of the stripe and repeats the process. With this greedy algorithm, the suggested system can generate ASCII art in a short period of time using a multi-threaded implementation. Since the suggested algorithm does not use a neural network or machine learning, it can immediately respond to changes in font type or font size, or changes in the type or number of characters used.

The results generated from the suggested algorithm were compared with a state-of-the-art ASCII art generation system [5] and the CNN-based system [6]. The results show that the suggested system can generate ASCII art at a much faster rate than the existing system. Our contributions can be summarized as follows:

- A novel character-image matching algorithm extracts orientation features from the image and finds the most suitable characters through comparison with such features.
- An efficient text placement algorithm places characters on the stripes of the image. This algorithm allows us to generate ASCII art from a 512×512 input image in up to 8.7 seconds.

II. RELATED WORKS

A. STRUCTURE-BASED ASCII ART SYNTHESIS

There have been several studies on ASCII art generating methods. Xu *et al.* [2] created ASCII art from an input image using the alignment-insensitive shape similarity (AISS) metric and constrained deformation model. Miyake *et al.* [3], proposed real-time ASCII art generation based on the glyph matching method using Normalized Cross-Correlation (NCC) or Histogram of Oriented Gradients (HOG). However, these methods can only be used in environments where fixed-width fonts are used: not in an environment using proportional fonts. To create ASCII art in a proportional font environment, a similarity metric and dynamic programming-based optimal placement method using multi-orientation phase congruency model is suggested (Xu *et al.* [4]). Xu *et al.* [5] further improved the output quality using non-classical receptive field (non-CRF) modulation to extract structures from an image to create ASCII art. The system can generate ASCII art from natural photographs as well as from line drawing images. However, it takes much time to create ASCII art of a decent quality with this system. To tackle this issue, we suggest a new similarity metric and a fast text placement method in this study.

Akiyama suggested an ASCII art generating method using a CNN [6]. This study used the ASCII art created by a human artist and a rough sketch converted from the ASCII art to train the network. Although this method was able to produce art of good quality, the network was not trained with hand-drawn original images but instead with system-generated images. Such a bias in the dataset may decrease the adaptability of the learned network, for example, when the original image has a dark background. Moreover, only particular fonts-based ASCII art can be generated.

B. STRUCTURE LINE DETECTION

To produce structure-based ASCII art, it is necessary to detect the structure lines from the input image. One of the simplest ways of structure line detection is to use an edge detector such as the canny edge detector [7]. However, edge detectors depend largely on the contrast and scale of the image. Several studies have proposed methods to extract structure lines from input images. Kang *et al.* [8] used a flow-based difference of gaussian (DoG) filter to create a line drawing style image from an input image. Arbelaez *et al.* [9], proposed a contour detector that combines multiple local cues with a spectral clustering-based globalization framework. Kokkinos [10] proposed a boundary detection system using a deep CNN. Simo-Serra [11] suggested a CNN-based framework that converts a pencil drawn rough sketch to a clean line drawing image. Li *et al.* [12], proposed a deep network model that can remove the texture from a manga image with screen patterns and extract structural lines from it. This study used the method employed by Kang *et al.* [8] in generating a structure line image from an original image.

C. GENERATING FEATURES

The structure line detection algorithm can extract the structure lines of the original image, but it is difficult to use those lines for ASCII art synthesis. The thinning algorithm can be used to facilitate the comparison between structure lines and ASCII characters. Studies on thinning have been around for several decades, with a variety of thinning algorithms being proposed [13]–[16].

An image similarity metric can be used in matching the structure lines with the ASCII characters. For example, SSIM

and its extensions compare the luminance, contrast, and structure of an image to calculate the similarity [17]–[20]. Meanwhile, a phase congruency-based similarity metric has also been proposed [21]. Phase congruency is a technique used for the state-of-the-art technique of ASCII art synthesis [4], [5]. The SIFT [22] identifies key points from gaussian differences of an image for image matching. In this study we used an image's pixel orientation-based similarity metric [23]. This method can identify the orientation of the pixels with accuracy using the surrounding pixels' data, and can be applied to fingerprint recognition, etc [24].

D. IMAGE STYLIZATION

Research regarding image stylization, which translates input images into different styles of images, has been conducted for several years and considerable advances have been achieved. Decarlo and Santella [25] converted an input image to a line-drawing style image through image segmentation and edge detection, while Lu *et al.* [26] generated a pencil drawing style image from a natural image. Kim *et al.* [27] presented a GAN-based system that uses color tag information to paint the image, Chen *et al.* [28] suggested a fully convolutional network to perform various advanced image processing operations, Fischer *et al.* [29] presented stylization of an augmented reality screen, and Lin *et al.* [30] proposed an abstraction layout that generates a flat design style black and white image in the 3D model. However, few studies on image hatching into hundreds of image patches, such as ASCII art synthesis, have been reported. This study aims to divide the input image into various types of rectangular character images.

III. IMAGE PREPROCESSING

A. STRUCTURE LINE EXTRACTION

In general, the font characters used for ASCII art are a one-pixel-width binary image. However, most of the input images are not binary images. Therefore, it is necessary to extract the structure lines from the input image and convert those to one-pixel-width binary images. In this study, the method employed by Kang *et al.* [8] was used in extracting structure lines from the input image. Using this method, the edge tangent flow and flow-based DoG filter can be used to generate a line drawing style image preserving the structure of the input image. For this study, the parameters were adjusted properly to generate the binary image.

B. THINNING

Since the structure line image is not a one-pixel-width image, a thinning operation is required to make it easier to comparing with font characters. However, before the thinning operation, the noise of the structure lines should first be removed. We used the pre-thinning method employed by Dong *et al.* [14] for this de-noising operation. Under this pre-thinning method, the value of the p changes according to the value of its eight neighboring pixels (see Figure 2). The p value

p_4	p_3	p_2
p_5	p	p_1
p_6	p_7	p_8

FIGURE 2. Labelling 8 neighboring pixels of p.

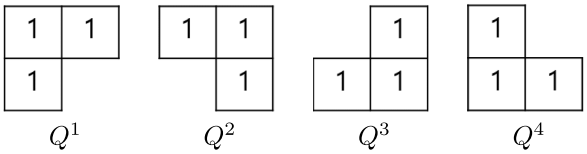


FIGURE 3. Q^k template for m_t score calculation ($k \in [1..4]$).

changes as follows:

$$B_{odd}(p) = p_1 + p_3 + p_5 + p_7. \tag{1}$$

$$p' = \begin{cases} 0 & (B_{odd}(p) < 2) \\ 1 & (B_{odd}(p) > 2) \\ p & otherwise \end{cases} \tag{2}$$

Jang and chin [31] suggested m_t scores to measure the convergence of a thinned image S_m. m_t can be defined as follows:

$$m_t = 1 - \left(\frac{Area\left[\bigcup_{k\in[1..4]} S_m Q^k\right]}{Area\left[S_m\right]} \right), \tag{3}$$

where $Area[]$ counts the number of one-pixels of the skeleton and Q^k is the pattern given in Figure 3. This means that the fewer Q^k patterns that S_m has, the closer the S_m is to the perfect unit skeleton. In this study, KMM thinning [15] was used since the results of the thinning had fewer Q^k patterns. Figure 4 shows the results of several thinning algorithms.

IV. OVERVIEW

Our system generates ASCII art from the input image of a one-pixel-width binary image. To generate ASCII art, the suggested system uses additional character data. The system can generate ASCII art from various proportional fonts. In our lab environment, the font chosen for ASCII art was Saitamaar [32] of 16px height, which is a font frequently used for displaying ASCII arts.

As illustrated in Figure 5 below, the suggested system consists of five algorithms. In the feature extraction step, the local orientation of each pixel is calculated from the input image. The stripe segmentation step divides the input image and features into stripes of specified pixel height. The character score estimation step calculates the character score for each location of the stripes, and in the text placement step, character scores are used to place a character on the stripes. Finally, the ASCII art completion step combines all the text stripes to create one ASCII art. Details of each step are described in Section 5 below.

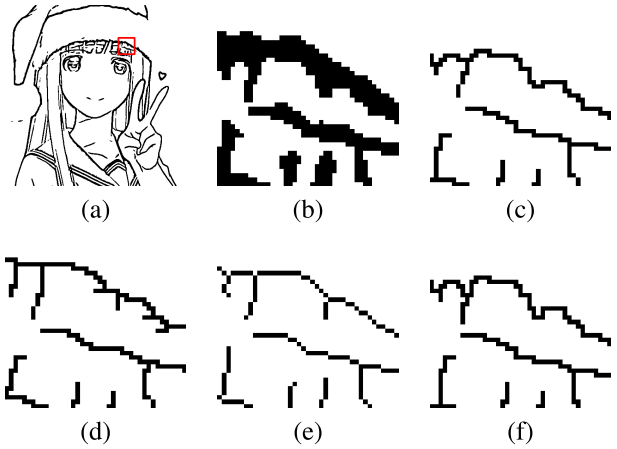


FIGURE 4. Various thinning operation results: (a) Original (b) Red box of original (c) Zhang and Suen [13] (d) Dong, *et al.* [14] (e) KMM [15] (f) K3M [16].

V. ALGORITHM

This section covers the algorithms of the suggested system in detail. Section 5.1 describes the feature extraction, Section 5.2 the character score estimation, and Section 5.3 the text placement. The stripe segmentation and ASCII art completion step simply divides the image and combines it again.

A. FEATURE EXTRACTION

To create ASCII art, a human artist compares an original image with a character image. To mimic a human artist, the suggested system extracts features from the input image. The state-of-the-art technique for generating ASCII art uses a 6-way phase congruency using these features [5]. For sophisticated character score calculations, the suggested system calculates pixel orientation.

We used the method suggested in [23] for pixel orientation calculation. First, gaussian blur was applied to the input image I. We created a blurred image I' using a gaussian blur filter. We set parameters of the gaussian filter as 3×3 size and $\sigma = 0.7$ through experiments. The local orientation of the pixel p on the blurred image I' is as follows:

$$V_x(p) = \sum_{(u,v)\in W} {G_x}^2(u,v) - {G_y}^2(u,v), \tag{4}$$

$$V_y(p) = \sum_{(u,v)\in W} 2G_x(u,v)\,G_y(u,v), \tag{5}$$

$$\theta'(p) = \frac{1}{2} tan^{-1}\left(\frac{V_y(p)}{V_x(p)}\right), \tag{6}$$

$$\theta(p) = \begin{cases} \theta'(p) + \dfrac{\pi}{2}, & (V_x(p) \geq 0) \\ \theta'(p), & (otherwise) \end{cases} \tag{7}$$

where G_x and G_y are the gradients in the x-axis and y-axis direction of the image and W is an image block containing p. In this study, we used the scharr filter as a gradient value and W was set as a square block of 5×5 centered around p. Figure 5 (b) shows an example of the visualized pixel orientation of the input image. Pixel orientation for every pixel in I' was calculated.

B. CHARACTER SCORE ESTIMATION

For text placement, the character score measures the similarity between the character and the image on top of the image stripes. If the character and the image are perfectly matched, the pixel value and the pixel orientation value will be identical. The character match scores can be calculated based on this idea; however, the pixel orientation can be defined only around the foreground pixels, not among the background pixels. Therefore, this study defined the character match score $S_m(c)$ only when the pixels p_c of the character image c are foreground pixels. $S_m(c)$ is defined as follows:

$$S_m(c) = \sum_{p_c \in W_c} S_m(p_c, c), \tag{8}$$

$$S_m(p_c, c) = \begin{cases} 0, & (p_c \notin F_c) \\ 0, & (p_c \notin T_c) \\ \Delta p\,(cos(\Delta\theta) + 1), & (otherwise) \end{cases} \tag{9}$$

$$\Delta p = \left|p'_i - p'_c\right|, \tag{10}$$

$$\Delta\theta = \left|\theta\left(p'_i\right) - \theta\left(p'_c\right)\right|, \tag{11}$$

where p'_i and p'_c are a blurred pixel value of the input image and the character image, respectively, W_c is a character image area, F_c is a set of foreground pixels on c, and T_c a set of pixels on c having a valid $\Delta\theta(p)$ value.

However, if only the character match score is used, other characters that are not suitable may be matched due to the overmatched score. As such, we designed the algorithm to prevent overmatching by introducing mismatch scores. As the difference between the character pixel value and image pixel value becomes greater, as well as that between the character pixel orientation and image pixel orientation, the mismatch score becomes higher. As with the match score, for the mismatch score, p_c is always 1 because p_c is defined only when among the foreground pixels. Therefore, the mismatch score $S_u(c)$ and character score $S(c)$ are defined as follows:

$$S(c) = \omega_u S_u(c) - \omega_m S_m(c), \tag{12}$$

$$S_u(c) = \sum_{p_c \in W_c} S_u(p_c, c), \tag{13}$$

$$S_u(p_c, c) = \begin{cases} 0, & (p_i \notin F_c) \\ 2 - p'_i, & (p_c \notin T_c) \\ 1 - p'_i + sin(\Delta\theta). & (otherwise) \end{cases} \tag{14}$$

Here, ω_u and ω_m are weight parameters. The user can easily control the style of ASCII art by changing these parameters. A wide range of weighting parameters works reasonably well. We set the weight values as $\omega_u = 0.65$ and $\omega_m = 1$ through experiments.

C. TEXT PLACEMENT

Placing the text on the image stripes can be a true challenge. One simple way is to place the characters, one-by-one, from

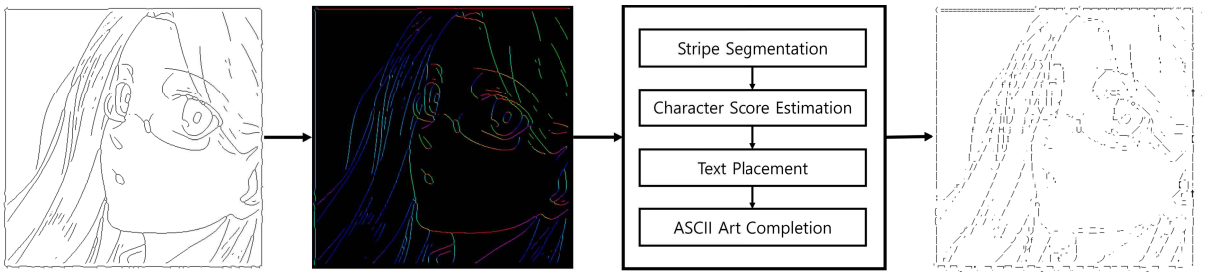


FIGURE 5. Overview of our system.

the left to the right of the stripes. This method is intuitive and simple, but quality may degrade due to differences in character widths. In order to create ASCII art in a proportional font environment, preceding studies suggest character placement schemes which use dynamic programming [4], [5]. A dynamic programming-based scheme is used for obtaining an optimal solution of the specific quality objective of the ASCII art [4], [5]; however, this particular class of dynamic programming algorithms cannot easily be implemented using parallel processing due to data dependencies between sub-problems.

Therefore, this study proposes a stripe-based text placement algorithm that can generate ASCII art faster through parallelization. Our algorithm creates ASCII art text stripes from the stripes of the input image. The stripes of the input image are obtained by slicing the input image horizontally into several small images with width w and height h, where w is the width of the input image, and $h(= 16px)$ is the font-height in pixel units. The number of stripes N is obtained by dividing the height of the input image by a stride value $(= 18px)$ slightly higher than the font-height h to express gaps between text lines.

The text placement algorithm approach works similar to the divide and conquer process. To create text stripes, the character score table calculated in Section 5.2 was used as input values. For the input score table $S[0, W]$, the system divides the table into three subsequences: $S[0, l]$, C, and $S[r, W]$. For each subsequence, text stripes were then repeatedly created, where W is each input stripe width, C is the character with the smallest character score in the score table, and l and r, are the left and right positions of C, respectively. If C is a space character, the system checks whether S can be filled with only space characters, instead of dividing S into three subsequences.

During the repetitive operations, the system may not be able to generate text stripes for the subsequence S. For example, if the width of the S is 2px and the minimum width of the C is 3px, the text stripes for S cannot be created. In this case, the system generates text stripes for another subsequence that contains the second smaller character C' instead of C of the original subsequence. Since the right-side boundary of the ASCII art is often not perfectly aligned, it is assumed that if the input subsequence contains the right-side boundary of the original sequence, the text stripes can always be created. Algorithm 1 is the pseudocode of the algorithm proposed in this study.

Figure 6 compares the results of left-to-right text placement with that of the proposed algorithm. For the left-to-right algorithm, the structure of the input image may not be reflected to the same degree as achieved with the suggested algorithm. For example, if there is a space on the left side of the image, the ASCII art created will have a discrepancy. The proposed algorithm, however, places the characters with a well-retained structure first owing to the use of character scores; hence, no discrepancy will be found in the generated ASCII art.

VI. RESULTS

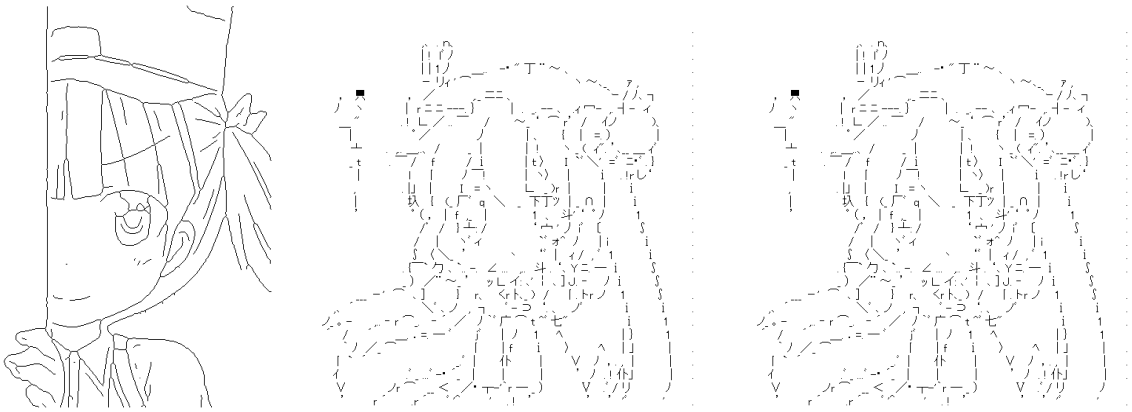
The suggested system in this study generates ASCII art from a one-pixel-width binary image. As the input images, the Danbooru dataset [33] and Akiyama's input data [6] were used. The original image was converted to a one-pixel-width binary image via the preprocessing described in Section 3. A total of 752 characters of 16px height Saitamaar font [32] were used to create ASCII art. Figure 7 below shows the original image, the preprocessed image, and the ASCII art generated by the suggested system.

A. PERFORMANCE

The suggested ASCII art generating system was built using C++, OpenCV library [34]. Our lab environment was a PC with an AMD Ryzen 7 3700MHz, 32GB main memory, and NVIDIA Geforce GTX 1080Ti. The system performance was measured by randomly selecting images with a 512 × 512 pixel resolution. Table 1 shows the runtime for each step of the system. The total runtime of the system is shown in Table 4. The suggested system can generate ASCII art from the input image in an average of 45.5 seconds under a single-threaded environment and 10.3 seconds under a multi-threaded environment.

B. COMPARISON

This section compares our suggested system with the existing, state-of-the-art ASCII art generation algorithm. For comparison, the system proposed by Xu, *et al.*, [5] and



(a) (b) (c)

FIGURE 6. Comparison of the left-to-right text placement method and proposed algorithms: (a) Input (b) Left-to-right (c) Proposed algorithm.

Algorithm 1 Overview of Proposed Text Placement Algorithm

Input: character score table $S[0, W]$, boundary boolean b
Output: text stripe T
Initialization:
$b = true$,
$B = \{\text{`}\ \text{'}, \text{`}\ \text{'}\}$. (half-width and full-width white-space characters)

function $GenAA(S[0, W], b)$
  $C = min_{score}(S[0, W])$
  while $C \notin B$ **do**
    $l, r = location(C)$
    $L = GenAA(S[0, l], false)$
    $R = GenAA(S[r, W], b)$
    $l_f = Filled(L)$
    $r_f = Filled(R) \vee b$
    if $l_f \wedge r_f$ **then**
      $T = L \oplus C \oplus R$
      return T
    else
      $C = nextchar_{score}(S[0, W])$
    end if
  end while
  $T = FillBlank(S[0, W])$
  return T
end function

Akiyama [6] (as the state-of-the-art process) were used. Even though it was not possible to obtain the source code or executable program for Xu *et al.*'s system, a brief performance summary for the system was available. For Akiyama's system, we ran the program on a PC having the same lab environment as ours.

The performance of the Xu *et al.* system was measured on a PC with a 2GHz CPU, and 8GB memory. Table 2 compares the performance of the suggested system with that of Xu *et al.*

TABLE 1. Execution time (milliseconds) of the proposed system by stage: (a) Feature extraction (b) Stripe segmentation (c) Character score estimation (d) Text placement (e) ASCII art completion.

Image No.	step (a)	step (b)	step (c)	step (d)	step (e)
A	47	48	42307	2165	0.192
B	62	31	41847	3294	0.155
C	62	47	42747	3190	0.199
D	47	32	42389	2501	0.191
E	47	47	43779	3064	0.229

TABLE 2. Timing statistics and comparison with the Xu *et al.* system.

Image No.	CPU performance	number of characters	execution time
A	3.7GHz	752	177s
B	3.7GHz	752	175s
C	3.7GHz	752	189s
D	3.7GHz	752	175s
E	3.7GHz	752	178s
Average	3.7GHz	752	178.8s
Xu et al. [5]	2.0GHz	305	18min

1024×1024 pixel resolution was used for the input image in this comparison. Although the CPU clock performance under our lab environment was only 1.85 times greater than that of Xu *et al.*, the suggested system was about six times faster, even with more than 2.4 times more character types.

Figure 8, Table 3, and Table 4 show the comparison between Akiyama's system and our suggested system. Akiyama employed a system that uses a CNN to create ASCII art. It generates several ASCII artworks by moving the original image by one pixel. However, in the suggested system, the original image is fixed and only one ASCII art is generated. An image with a 512×512 pixel resolution was used as the input image. According to the results, our system obtained better Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity (SSIM) scores than the Akiyama's system. This indicates that our system can preserve both the content and the structure of the input image better than the

FIGURE 7. Results of the proposed algorithm: (a) Original [33] (b) Structural lines obtained using Kang, *et al.* [8] (c) Thinned structural lines (d) Generated ASCII art.

existing system. However, we also notice that the Akiyama's system reserves the thin lines of the input image better than our system in some parts of the image, although this was not quantitatively reflected in the scores. Since the Akiyama's system is a CNN-based algorithm, it relies on many examples and additional training procedures are required if the

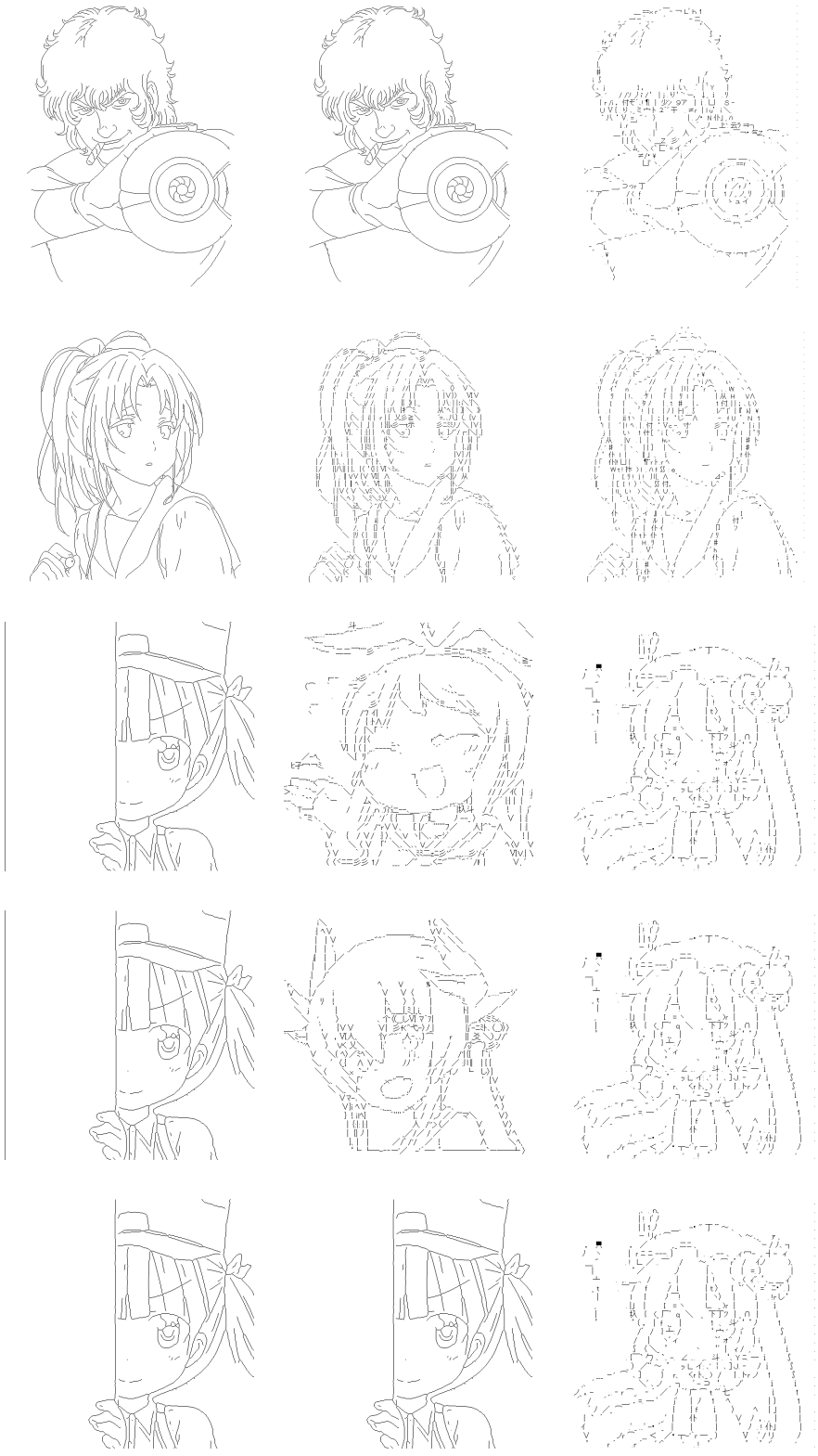
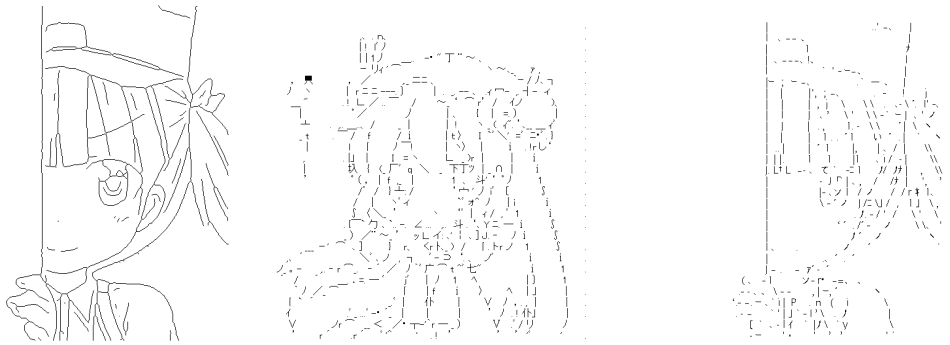


FIGURE 8. Comparison with the CNN-based method: (a) Input (b) Akiyama's [6] (c) Ours.

execution environment changes (e.g., when using a different font). Our suggested system can respond immediately to changes in the execution environment. Our system also runs faster; in particular, in a multi-threading environment, ASCII



(a) Original (b) Saitamaar (c) Afta Sans

FIGURE 9. Generated ASCII art with two different fonts.

TABLE 3. PSNR and SSIM score comparison with Akiyama's system.

Image No.	PSNR (Ours)	PSNR (Akiyama)	SSIM (Ours)	SSIM (Akiyama)
A	16.749	14.597	0.830459	0.766226
B	15.0901	12.8858	0.747789	0.639855
C	15.6045	13.7829	0.777222	0.70902
D	13.6293	11.7029	0.634619	0.53244
E	15.5199	13.6586	0.782515	0.717454

TABLE 4. Timing statistics and comparison with Akiyama's system.

Image No.	execution time (single thread)	execution time (multi threads)	execution time (Akiyama)
A	44.6s	8.7s	49.9s
B	45.2s	11.4s	46.9s
C	46.0s	11.9s	49.1s
D	45.0s	10.2s	49.7s
E	46.9s	9.3s	52.2s
Average	45.5s	10.3s	49.6s

art can be generated within 12 seconds. This would allow human artists to make additional edits in a short time.

C. USING DIFFERENT FONTS

In this section, we experimentally show the robustness of the proposed algorithm using two different fonts (the Satiamaar and Afta Sans fonts). The Afta Sans font environment uses 380 characters of 16px height. Figure 9 shows the original image, an ASCII image created with Saitamaar font, and an ASCII image created with Afta sans font.

VII. CONCLUSION AND LIMITATION

In this paper, we propose a new method for generating ASCII art in an environment using proportional fonts. Our method generates ASCII art using a character-image matching algorithm based on pixel orientation feature and a greedy text placement algorithm. The proposed method can generate ASCII art from an input image in a short time, regardless of the type of font or the character set. In particular, our algorithm can generate high quality ASCII art from images with many vertical lines or straight lines.

(a) input (b) ASCII Art

FIGURE 10. Example of a failed attempt to create ASCII art for a human face.

In addition, our experiments show that ASCII art can be generated in an average of 10.3 seconds from an image with a resolution of 512 × 512 pixels in an environment using multithreading. This timing performance helps the artist to quickly obtain a high-quality result, allowing the user to add additional texts or highlighting effects on the fly.

A. DISCUSSION

Using different kinds of characters can improve the quality of ASCII art. Ideally, the best quality would be achieved when using many kinds of unicode characters. However, using too many typefaces can adversely affect the time it takes to compare them with the input image. Using several types of similarly shaped characters may bring insignificant improvement in quality compared to increased calculation time. Also, some font types do not contain some characters. These constraints must be considered when determining the character set to be used in the creation of ASCII art.

B. LIMITATIONS AND FUTURE WORK

Our algorithm expresses the linear structure of an image well; however, there are structures that the algorithm does not handle well. For example, detailed and complex curved structures, such as human eyes, are particularly difficult to express well. Figure 10 shows the structure line input and the

generated ASCII art for an actual human face. The algorithm expressed the overall structure of the input image relatively well, but some facial features were missed.

Also, our current system cannot express the tone of the image. In the case of an artist, ASCII art often includes not only the structure of the image, but also the tone of the image. Developing a scheme for subtle tonal representation would be an interesting future research topic.

Another possible topic for future research is the development of real-time creation and authoring tools that can reflect user constraints. For example, an artist may want to put together several ASCII art pieces in an image, or pre-arrange certain characters in a specific location as a guide. These authoring tools can enable users to obtain desired results more quickly and intuitively.

We are also interested in combining the advantages of machine-learning-based methods. There have already been studies related to CNN-based ASCII art creation [6]. Methods, such as online transfer learning, may be able to quickly produce a network applicable to a new font-set.

REFERENCES

[1] N. Markuš, M. Fratarcangeli, I. S. Pandžić, and J. Ahlberg, "Fast rendering of image mosaics and ASCII art," *Comput. Graph. Forum*, vol. 34, no. 6, pp. 251–261, Sep. 2015.
[2] X. Xu, L. Zhang, and T.-T. Wong, "Structure-based ASCII art," *ACM Trans. Graph.*, vol. 29, no. 4, pp. 1–10, Jul. 2010.
[3] K. Miyake, H. Johan, and T. Nishita, "An interactive system for structure-based ascii art creation," in *Proc. NICOGRAPH Int.*, 2011, pp. 3–4.
[4] X. Xu, L. Zhong, M. Xie, J. Qin, Y. Chen, Q. Jin, T.-T. Wong, and G. Han, "Texture-aware ASCII art synthesis with proportional fonts," in *Proc. Workshop Non-Photorealistic Animation Rendering (NPAR)*. Istanbul, Turkey: Eurographics Association, 2015, pp. 183–193.
[5] X. Xu, L. Zhong, M. Xie, X. Liu, J. Qin, and T.-T. Wong, "ASCII art synthesis from natural photographs," *IEEE Trans. Vis. Comput. Graphics*, vol. 23, no. 8, pp. 1910–1923, Aug. 2017.
[6] O. Akiyama, "ASCII art synthesis with convolutional networks," in *Proc. NIPS Workshop Mach. Learn. Creativity Design*, 2017, pp. 1–7.
[7] J. Canny, "A computational approach to edge detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-8, no. 6, pp. 679–698, Nov. 1986.
[8] H. Kang, S. Lee, and C. K. Chui, "Coherent line drawing," in *Proc. 5th Int. Symp. Non-Photorealistic Animation Rendering (NPAR)*. New York, NY, USA: Association for Computing Machinery, 2007, pp. 43–50.
[9] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik, "Contour detection and hierarchical image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 5, pp. 898–916, May 2011.
[10] I. Kokkinos, "Pushing the boundaries of boundary detection using deep learning," 2015, *arXiv:1511.07386*.
[11] E. Simo-Serra, S. Iizuka, and H. Ishikawa, "Mastering sketching: Adversarial augmentation for structured prediction," *ACM Trans. Graph.*, vol. 37, no. 1, pp. 1–13, Feb. 2018.
[12] C. Li, X. Liu, and T.-T. Wong, "Deep extraction of Manga structural lines," *ACM Trans. Graph.*, vol. 36, no. 4, pp. 1–12, Jul. 2017.
[13] T. Y. Zhang and C. Y. Suen, "A fast parallel algorithm for thinning digital patterns," *Commun. ACM*, vol. 27, no. 3, pp. 236–239, Mar. 1984.
[14] J. Dong, Y. Chen, Z. Yang, and B. W.-K. Ling, "A parallel thinning algorithm based on stroke continuity detection," *Signal, Image Video Process.*, vol. 11, no. 5, pp. 873–879, Jul. 2017.
[15] K. Saeed, M. Rybnik, and M. Tabedzki, "Implementation and advanced results on the non-interrupted skeletonization algorithm," in *Computer Analysis of Images and Patterns*, W. Skarbek, Ed. Berlin, Germany: Springer, 2001, pp. 601–609.
[16] K. Saeed, M. Tab dzki, M. Rybnik, and M. Adamski, "K3M: A universal algorithm for image skeletonization and a review of thinning techniques," *Int. J. Appl. Math. Comput. Sci.*, vol. 20, no. 2, pp. 317–335, Jun. 2010.
[17] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Trans. Image Process.*, vol. 13, no. 4, pp. 600–612, Apr. 2004.
[18] Z. Wang, E. Simoncelli, and A. Bovik, "Multiscale structural similarity for image quality assessment," in *Proc. 37th Asilomar Conf. Signals, Syst. Comput.*, vol. 2, 2003, pp. 1398–1402.
[19] M. P. Sampat, Z. Wang, S. Gupta, A. C. Bovik, and M. K. Markey, "Complex wavelet structural similarity: A new image similarity index," *IEEE Trans. Image Process.*, vol. 18, no. 11, pp. 2385–2401, Nov. 2009.
[20] Z. Wang and Q. Li, "Information content weighting for perceptual image quality assessment," *IEEE Trans. Image Process.*, vol. 20, no. 5, pp. 1185–1198, May 2011.
[21] L. Zhang, L. Zhang, X. Mou, and D. Zhang, "FSIM: A feature similarity index for image quality assessment," *IEEE Trans. Image Process.*, vol. 20, no. 8, pp. 2378–2386, Aug. 2011.
[22] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vis.*, vol. 60, no. 2, pp. 91–110, Nov. 2004.
[23] A. R. Rao and R. Jain, "A taxonomy for texture description and identification," Ph.D. dissertation, Univ. Michigan, Ann Arbor, MI, USA, 1989.
[24] A. K. Jain, L. Hong, S. Pankanti, and R. Bolle, "An identity-authentication system using fingerprints," *Proc. IEEE*, vol. 85, no. 9, pp. 1365–1388, Sep. 1997.
[25] D. Decarlo and A. Santella, "Stylization and abstraction of photographs," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 769–776, Jul. 2002.
[26] C. Lu, L. Xu, and J. Jia, "Combining sketch and tone for pencil drawing production," in *Proc. Symp. Non-Photorealistic Animation Rendering (NPAR)*. Annecy, France: Eurographics Association, 2012, pp. 65–73.
[27] H. Kim, H. Y. Jhoo, E. Park, and S. Yoo, "Tag2Pix: Line art colorization using text tag with SECat and changing loss," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 9055–9064.
[28] Q. Chen, J. Xu, and V. Koltun, "Fast image processing with fully-convolutional networks," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 2516–2525.
[29] J. Fischer, D. Bartz, and W. Strasser, "Stylized augmented reality for improved immersion," in *Proc. IEEE Virtual Reality Conf. (VR)*, Mar. 2005, pp. 195–202.
[30] Y.-E. Lin, Y.-L. Yang, and H.-K. Chu, "Scale-aware black-and-white abstraction of 3D shapes," *ACM Trans. Graph.*, vol. 37, no. 4, pp. 1–11, Aug. 2018.
[31] B. K. Jang and R. T. Chin, "One-pass parallel thinning: Analysis, properties, and quantitative evaluation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 14, no. 11, pp. 1129–1140, Nov. 1992.
[32] Accessed: Jun. 18, 2021. *Github—Keage/Saitamaar: AA Display Font Saitamaar Looks Like AA Script Applied to the Element*. [Online]. Available: https://github.com/keage/Saitamaar
[33] G. Branwen. (Jan. 2021). Danbooru2020: A large-scale crowdsourced and tagged anime illustration dataset. Anonymous and Danbooru Community. [Online]. Available: https://www.gwern.net/Danbooru2020
[34] G. Bradski and A. Kaehler, "OpenCV," *Dr. Dobb's J. Softw. Tools*, vol. 3, p. 2, Nov. 2000.

MOONJUN CHUNG received the B.S. degree in computer science from Hanyang University, where he is currently pursuing the master's and Ph.D. degrees in computer software. His research interests include computer graphics and image processing.

TAESOO KWON received the Ph.D. degree from the Department of Computer Science, Korea Institute of Science and Technology, in 2007. He is currently with the Department of Computer Software Engineering, Hanyang University, Seoul. His research interests include computer animation, physics-based character animation, and machine learning.