The International Journal of Parallel, Emergent and Distributed Systems
Vol. 00, No. 00, Month 2011, 1–15

RESEARCH ARTICLE

A Character Art Generator using the Local Exhaustive Search, with GPU acceleration

Yuji Takeuchi*, Koji Nakano[a]*, Daisuke Takafuji*, Yasuaki Ito*

[a]*Department of Information Engineering, Hiroshima University*
Kagamiyama 1-4-1, Higashi Hiroshima, 739-8527 Japan

(*Received 00 Month 200x; in final form 00 Month 200x*)

An ASCII art is a matrix of ASCII code characters that reproduces an original gray-scale image. A JIS art is an ASCII art that uses JIS Kanji code characters instead of ASCII code characters. They are commonly used to represent pseudo gray-scale images in text based messages. Since automatic generation of high quality ASCII/JIS art images is very hard, they are usually produced by hand. The main contribution of this paper is to propose a new technique to generate an ASCII/JIS art that reproduces the original tone and the details of an input gray-scale image. Our new technique is inspired by the local exhaustive search to optimize binary images for printing based on the characteristic of the human visual system. Although it can generate high quality ASCII/JIS art images, a lot of computing time is necessary for the local exhaustive search. Hence, we have implemented our new technique in a GPU to accelerate the computation. The experimental results shows that the GPU implementation can achieve a speedup factor up to 89.56 over the conventional CPU implementation.

Keywords: ASCII art, local exhaustive search, human visual system, GPU, parallel computing

1. Introduction

An ASCII art is a matrix of ASCII code characters reproducing an original image. *A JIS art* is an ASCII art that uses JIS Kanji code characters instead of ASCII code characters. ASCII/JIS arts are commonly used to show pseudo gray-scale images on devices or environment that can only display characters. ASCII arts have a long history, and exist before the computers have been developed. One of the most famous examples of ASCII arts represents the tail of a rat, published in "Alice's Adventures in Wonderland" [1]. As Internet becomes popular, ASCII arts have been used in various situations, such as the contents of e-mails and bulletin boards on the Web. The main purpose of treating ASCII arts is to print easier, or to communicate as alternative of graphics in the situations which the communication of graphics is impossible.

ASCII/JIS arts can be roughly classified into two major categories: the tone-based ASCII art and the structure-based ASCII art [2]. In the tone-based ASCII art, an original gray-scale image is converted into a matrix of characters so that the intensity level is reproduced (Fig. 2). Usually, the original gray-scale image is partitioned into blocks of a character size, and a character is assigned to each block such that the intensity level is preserved. On the other hand, the structure-based ASCII art is generated by converting an original gray-scale image into a matrix

*Corresponding author. Email: nakano@cs.hiroshima-u.ac.jp

ISSN: 1744-5760 print/ISSN 1744-5779 online
© 2011 Taylor & Francis
DOI: 10.1080/17445760.YYYY.CATSid
http://www.informaworld.com

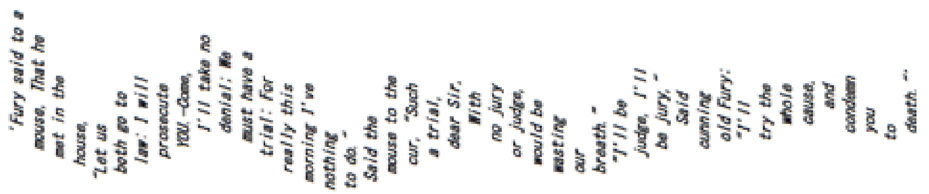


Figure 1. ASCII art representing the tail of a rat (from [1])

of characters so that the shapes of the original image is reproduced (Fig. 3). A character is assigned to each block such that the shape of the block is preserved.

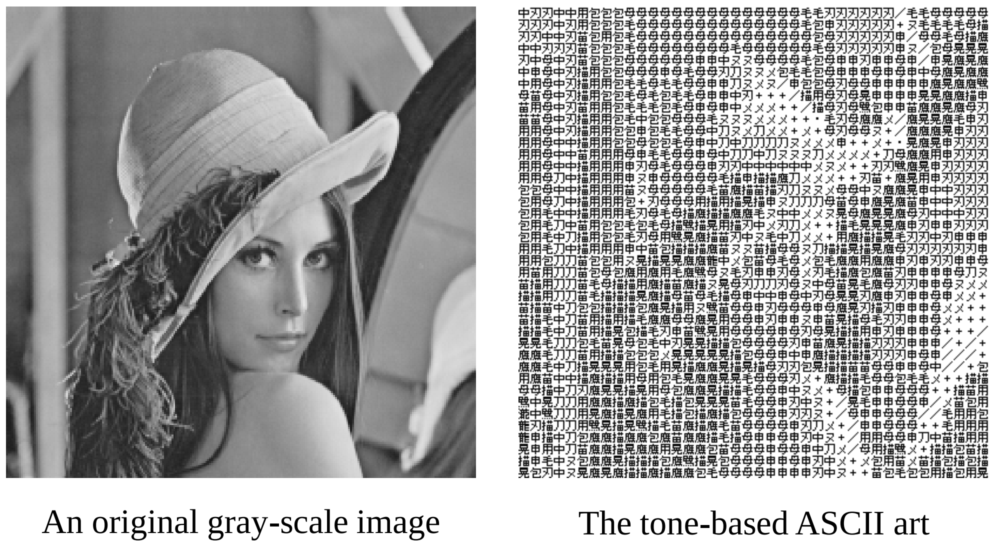


Figure 2. The tone-based ASCII art

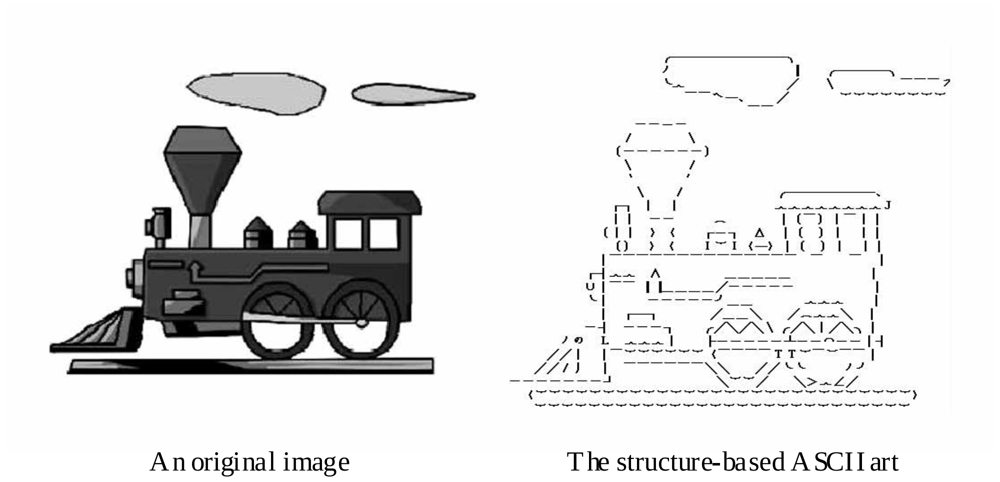


Figure 3. The structure-based ASCII art (from [2])

The main contribution of this paper is to propose a new method for generating ASCII/JIS art images which can maintain the smooth changes of intensity levels and the shapes in an original gray-scale images. The resulting ASCII/JIS art by our method is essentially the tone-based ASCII art, but it also has a flavor of the structure-based ASCII art. Our new approach is inspired by digital halftoning [3, 4] of gray-scale images into binary images for printing. In particular, it uses a technique of the local exhaustive search [5, 6] for digital halftoning, which can generate a binary image that preserves the details and the intensity levels of an original input gray-scale image. It is known that the direct binary search [7] can generate high quality binary images that reproduces the details and the tones of original gray-scale images. Later, the direct binary search is extended to the local exhaustive search [5, 6], which can generate better binary images. Our new method for ASCII/JIS art generation uses the local exhaustive search, and can reproduces the details and the tones of original gray-scale images.

In a conventional method for generating a tone-based ASCII art, a character is selected for each block of an original image such that the average intensity level is preserved. In other words, a character with the most similar intensity level of the corresponding block in an original image is selected. For example, a free software "Text artist" [8] uses this approach. Though this method is very simple and can be implemented easily, the details and the intensity level of an original image is not reproduced well. In [9], intensity level of an original image is reproduced by adjusting the space of characters. However, the details of the original image are not reproduced. In [10], ASCII art generation for original binary images was shown. This method works well for binary images, but cannot handle gray-scale images.

Our new approach first initializes a matrix of characters by the conventional tone-based ASCII art generation. After that, characters are repeatedly replaced by the best character among all available characters. To select the best character, a matrix of characters is blurred using the Gaussian filter and the pixel-wise difference of the blurred image and the original image is computed as an error. The best character is selected so that the total error is minimized. This replacement is repeated until no more improvement is possible. The resulting matrix of characters reproduces the original gray-scale image very well, because the error of the blurred matrix of characters and the original gray-scale image is small and the Gaussian filter approximates the human visual system. However, compared with a known approach, our approach requires enormous amount of computation to search the best character image among all characters.

The GPU (Graphics Processing Unit), is a specialized circuit designed to accelerate computation for building and manipulating images [11–15]. Although GPUs are optimized and designed for graphics computing, latest GPUs support more general operations for general purpose computing. Hence, GPUs have recently attracted the attention of many application developers [11, 16–19]. NVIDIA provides a parallel computing architecture called *CUDA* (Compute Unified Device Architecture) [20], the computing engine for NVIDIA GPUs. CUDA gives developers access to the virtual instruction set and memory of the parallel computational elements in NVIDIA GPUs. In many cases, GPUs are more efficient than multicore processors [12], since they have hundreds of processor cores and very high memory bandwidth. To accelerate our new approach, we have parallelized the replacing process so that the replacement is performed for multiple blocks in parallel. We have implemented our method in a CUDA-enabled GPU and evaluated the performance on NVIDIA GeForce GTX 780 Ti. For ASCII art generation for an original image of size 1024×1024 using 95 ASCII code characters, our GPU implementation runs in 0.0517s, while the Intel CPU (Xeon E5-2430, 2.2GHz) implementation runs in 2.952s. Further, if we use 7310 JIS Kanji code characters, our GPU implementation for JIS art runs in only 0.7149s, while the CPU implementation runs in 64.03s. If this is the case, the GPU implementation can achieve a speedup factor 89.56 over the conventional CPU implementation. In a preliminary version of this paper [21] presented as a conference paper, we have presented the basic idea of ASCII/JIS art generation using the local exhaustive search, and showed experimental results on

GeForce GTX 680. In this extended journal version, we optimize the search range, a parameter of the local exhaustive search, to balance the quality ASCII/JIS art images with the computing time on the latest GPU, GeForce GTX 780 Ti.

This paper is organized as follows. Section 2 explains a conventional method for generating the tone-based ASCII/JIS art. In Section 3, we show outline of our new method based on the local exhaustive search for the tone-based ASCII/JIS art. We then go on to show an algorithm and an implementation of our method for generating the tone-based ASCII/JIS art using the local exhaustive search in Section 4. In Section 5, we show how we have implemented our method in the GPU to accelerate the computation. Section 6 compares the resulting ASCII/JIS art images of the convention method and our method, and shows the computing time. Section 7 concludes our work.

2. A conventional method for the tone-based ASCII/JIS art generation

This section describes a conventional method for the tone-based ASCII/JIS art generation. The idea is to partition an original image into blocks of the same size as characters. Each block is assigned a character such that each character reproduces the intensity level of the corresponding block.

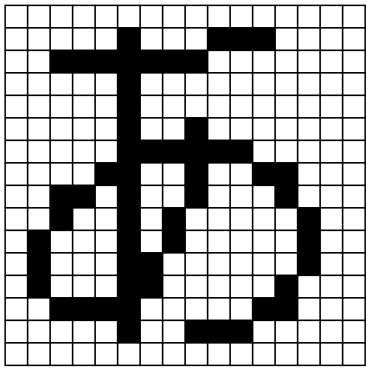


Figure 4. An example of the bitmap image of a character

Before showing the conventional algorithm, we review how each character is displayed as a bitmap image. Figure 4 shows an example of the bitmap image of a character. The bitmap image is a binary image with pixels 0 (black) or 1 (white). The bitmap image of Figure 4 is of size 16×16. It has 60 black pixels and 196 white pixels out of 256 pixels. Hence, we can think that the intensity level of the character is $\frac{196}{256} = 0.765625$. Let $c(i,j)$ $(0 \le i,j \le k-1)$ denote a pixel value (0 or 1) at position (i,j) of character c of bitmap size $k \times k$. We can compute the intensity level $I(c)$ of c as follows:

$$I(c) = \sum_{i=0}^{k-1} \sum_{j=0}^{k-1} \frac{c(i,j)}{k^2}.$$

Suppose that a gray-scale image $A = (a_{i,j})$ of size $n \times n$ is given, where $a_{i,j}$ denotes the intensity level at position $(i,j)(0 \le i,j \le n-1)$ taking a real value in the range $[0,1]$. The real value corresponds to the intensity level of each pixel, and 0 and 1 correspond to black and white, respectively. Let us partition the gray-scale image into $\frac{n}{k} \times \frac{n}{k}$ blocks of size $k \times k$ each. Let $A_{i',j'}$ $(0 \le i',j' \le \frac{n}{k} - 1)$ denote a block with k^2 pixels $a_{i,j}$ ($\frac{n}{k}\cdot i' \leq i \leq \frac{n}{k}\cdot(i'+1)-1, \frac{n}{k}\cdot j' \leq j \leq \frac{n}{k}\cdot(j'+1)-1$). It should be clear that the average intensity $I(A_{i',j'})$ of each block $A_{i',j'}$ is:

$$I(A_{i',j'}) = \sum_{i'=\frac{n}{k}\cdot i}^{\frac{n}{k}\cdot(i'+1)-1} \sum_{j'=\frac{n}{k}\cdot j}^{\frac{n}{k}\cdot(j'+1)-1} \frac{a_{i,j}}{k^2}. \tag{1}$$

Let C be a set of available characters. The conventional algorithm for the tone-based ASCII/JIS art image selects a character for each block such that the intensity level of a character is closest to the average intensity of the block. Let $B' = (b'_{i',j'})$ be an ASCII/JIS art such that each $b'_{i',j'}$ is a character in C. We determine each character $b_{i',j'}$ so that:

$$b'_{i',j'} = \arg\min_{c \in C} |I(A_{i',j'}) - I(c)|.$$

However, the distribution of the intensity levels of a character set C may be biased in the sense that it does not have characters with intensity levels close to 0 or 1. For example, a usual character set has no character with 1 white pixel and k^2-1 black pixels. Thus, the error $|I(A_{i',j'}) - I(c)|$ can be too large if $A_{i',j'}$ is close to 0 or 1. To resolve this problem, we adjust the intensity levels of an original image $A = (a_{i,j})$ as follows. Let H and L be the highest intensity level $\max\{I(c) \mid c \in C\}$ and and the lowest intensity level $\min\{I(c) \mid c \in C\}$, respectively. We adjust the intensity level of each pixel $a_{i,j}$ such that

$$a_{i,j} \leftarrow a_{i,j} \cdot (H - L) + L. \tag{2}$$

Clearly, the intensity level of each pixel takes a value in the range $[L, H]$, and thus, the average intensity level of each block $A_{i',j'}$ is always in $[L, H]$.

3. Our algorithm using the local exhaustive search

A new algorithm for generating an ASCII/JIS art using the local exhaustive search is presented in this section.

We use a Gaussian filter that approximates the characteristic of the human visual system. Let $G = (g_{p,q})$ denote a Gaussian filter, i.e. a 2-dimensional symmetric matrix of size $(2w+1) \times (2w+1)$, where each non-negative real number $g_{p,q}$ $(-w \leq p, q \leq w)$ is determined by a 2-dimensional Gaussian distribution such that their sum is 1. In other words,

$$g_{p,q} = s \cdot e^{-\frac{p^2+q^2}{2\sigma^2}} \tag{3}$$

where σ is a parameter of the Gaussian distribution and s is a fixed real number to satisfy $\sum_{-w \leq p,q \leq w} g_{p,q} = 1$.

Suppose that an ASCII/JIS art $B' = (b'_{i',j'})$ consists of $\frac{n}{k} \times \frac{n}{k}$ characters such that each $b'_{i',j'}$ is a character in C. Let $b'_{i',j'}(x, y)$ denote the intensity level of character $b'_{i',j'}$ at position (x, y). We can construct a binary image $B = (b_{i,j})$ of size $n \times n$ from B' as follows:

$$b_{i,j} = b'_{i/k, j/k}(i \bmod k, j \bmod k), \tag{4}$$

where $b_{i,j}$ is the intensity level of B at position (x, y). In other words, B is the resulting image obtained by rendering the ASCII/JIS art B'. We can obtain a blurred image $R = (r_{i,j})$ of B using the Gaussian filter G as follows:

$$r_{i,j} = \sum_{-w \le p,q \le w} g_{p,q} b_{i+p,j+q}$$

The idea of our ASCII/JIS art generation is to find an ASCII/JIS art B such that the blurred image R is very similar to the original image A. We define the error of R with respect to A as the sum of difference of the intensity levels as follows:

$$\mathrm{Error}(A, R) = \sum_{0 \le i,j \le n-1} |a_{i,j} - r_{i,j}|^2 \tag{5}$$

The goal of our method is to find the best ASCII/JIS art B^* so that

$$B^* = \arg\min_B \{\mathrm{Error}(A, R) \mid \\ B \text{ is an ASCII/JIS art using a character set } C\}.$$

Since it is a very hard problem to find the optimal ASCII/JIS art B^*, we use the approximation technique by the local exhaustive search. The outline of our algorithm that computes an ASCII/JIS art of an original gray-scale image A using a character set C is as follows:

[ASCII/JIS art generation by the local exhaustive search]

Step 1: Initialization

We generate an ASCII/JIS art B using the conventional algorithm for the tone-based ASCII/JIS art generation.

Step 2: The local exhaustive search

We pick an element $b'_{i',j'}$ in B' one by one from the top-left corner to the bottom-right corner in the raster scan order. We select a replacement character of $b'_{i',j'}$, which minimizes the total error over all characters in C, and replace $b'_{i',j'}$ by such c. This replacement procedure by the raster scan order is repeated until one round of raster scan order search from the top-left corner to the bottom-right corner does not replace characters and the error is not improved.

Step 3: Output

Compute a bitmap image B of the ASCII/JIS art B' and output it.

The reader should refer to Figure 5 illustrating the raster scan order local exhaustive search in Step 2. Note that this algorithm may not find the optimal ASCII/JIS art B^*. However, it can find a good approximation of the optimal ASCII/JIS art.

4. Implementation of ASCII/JIS art generation using the local exhaustive search

This section shows how each step of our new approach is implemented.

Again, let $k \times k$ be the size of characters in C. We can partition all characters in C into $k^2 + 1$ groups $C_0, C_1, \ldots, C_{k^2}$ such that each C_u $(0 \le u \le k^2)$ has characters with u white pixels and $k^2 - u$ black pixels. Clearly, the intensity levels of characters in C_u $(0 \le u \le k^2)$ is $\frac{u \cdot 1 + (k^2 - u) \cdot 0}{k^2} = \frac{u}{k^2}$. We assume that, for each character c in C, the blurred image c^g of the bitmap of c is computed in advance. The blurred

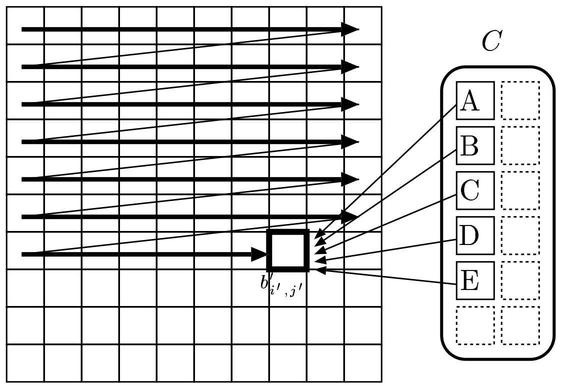


Figure 5. Step 2: the raster scan order local exhaustive search

image c^g has $(k+2w)\times(k+2w)$ pixels such that

$$c^g_{i,j} = \sum_{-w\le p,q\le w} g_{p,q}c_{i+p,j+q} \quad (-w \le i,j \le k+w-1).$$

In Step 1, we first adjust the intensity level of every pixel in an original gray-scale image $A=(a_{i,j})$ using formula (2). After that, we compute the average intensity level $I(A_{i',j'})$ of each block $A_{i',j'}$ using formula (1). For each block $A_{i',j'}$, we pick a character in C_u at random, where u satisfies

$$\frac{u-\frac{1}{2}}{k^2} \le I(A_{i',j'}) < \frac{u+\frac{1}{2}}{k^2}. \tag{6}$$

We can generate an ASCII/JIS art $B'=(b'_{i',j'})$ by choosing the picked character for $A_{i',j'}$ as a character of $b'_{i',j'}$. Also, from $B'=(b'_{i',j'})$, we can generate a bitmap image $B=(b_{i,j})$ by formula (4).

In Step 2, we first compute the blurred image $R=(r_{i,j})$ of the bitmap image $B=(b_{i,j})$ by computing formula (3). We compute the error matrix $E=(e_{i,j})$ such that

$$e_{i,j} = a_{i,j} - r_{i,j}.$$

Clearly, the total error is the sum of $|e_{i,j}|^2$ from formula (5). In Step 2, we need to find a replacement character c of $b'_{i',j'}$ that minimizes the total error. Clearly, it is sufficient to compute the total error of the affected region that includes the block $b'_{i',j'}$ as illustrated in Figure 6. The affected region is a region of the image B such that the Gaussian filter for the bitmap image of $b'_{i',j'}$ affects the pixel values of the blurred image. More specifically, the affected region of $b'_{i',j'}$ is a set $\mathcal{A}_{i',j'}$ of positions in the image such that

$$\mathcal{A}_{i',j'} = \{(i,j) \mid i'\cdot k - w \le i \le (i'+1)k+w-1, \\ j'\cdot k - w \le j \le (j'+1)k+w-1\}$$

Since the size of the Gaussian filter is $(2w+1)\times(2w+1)$, that of the affected region is $(k+2w)\times(k+2w)$. To find a replacement character c, we compute $e_{i,j} \leftarrow e_{i,j}+c^g_{i,j}$ in pixels in the affected region. Note that, after this computation, we can think that

$b'_{i',j'}$ is a character with each pixel having intensity level 0. After that, we compute the total error for each character c in C by evaluating the following formula:

$$\sum_{(i,j)\,\in\,\mathcal{A}_{i',j'}} |e_{i,j} - c^g_{i,j}|^2. \tag{7}$$

We evaluate this formula for all characters in C, and replace $b'_{i',j'}$ by c with the minimum total error. In other words, we execute the following operation:

$$b'_{i',j'} \leftarrow \arg\min_{c \in C} \sum_{(i,j)\,\in\,\mathcal{A}_{i',j'}} |e_{i,j} - c^g_{i,j}|^2. \tag{8}$$

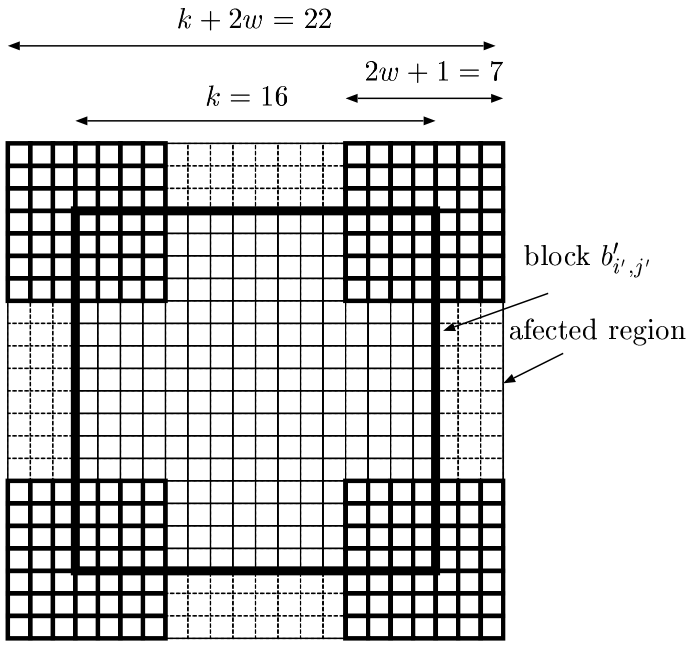
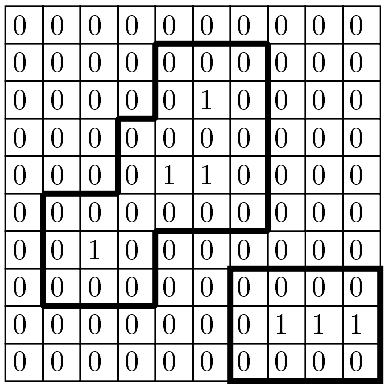


Figure 6. The affected region of a block $B'_{i',j'}$

To accelerate the local exhaustive search, we use two ideas: (1) replacement map, and (2) partial search. We first explain the idea of the replacement map. In Step 2, a round of the raster scan order search is repeated. It is possible that a region of an ASCII/JIS art is fixed in an earlier round, and no character in the region are not replaced until Step 2 terminates. Hence, it makes sense to perform the local exhaustive search for which characters might be replaced. For the purpose of determining if characters might be replaced, we use a replacement map $M = (m_{i',j'})$ of size $\frac{n}{k} \times \frac{n}{k}$. Before a round of the raster scan order search, all values in M is initialized by 0. We set $m_{i',j'} = 1$ if the operation in formula (8) replaces character $b'_{i',j'}$, that is, the right-hand side of formula (8) is not equal to $b'_{i',j'}$. Clearly, at the end of the round, $m_{i',j'} = 1$ if $b'_{i',j'}$ has been replaced in this round. Further, the affected region in which a character might be replaced in next round consists of (i', j') such that $m_{i',j'}$ or its neighbor takes value 1. Figure 7 illustrates an example of a replacement map and the affected region. In the next round, it is sufficient to perform the operation in formula (8) for the affected region.

The second idea, the partial search is used to reduce the computation of the right-hand side of formula (8). The intensity level of the right-hand side is close



0	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	1	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	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

Figure 7. The replacement map in an affected region

to $I(b'_{i',j'})$ with high probability, because it should be rare that the intensity level changes a lot by the local exhaustive search. Thus, it is not necessary to find the minimum over all characters in C. It is sufficient to evaluate the values of formula (7) for characters c in C such that $I(c)$ is close to $I(b'_{i',j'})$. More specifically, we perform the following operation:

$$b'_{i',j'} \leftarrow \arg\min_{c\in C'} \sum_{(i,j)\,\in\,\mathcal{A}_{i',j'}} |e_{i,j} - c^g_{i,j}|^2. \tag{9}$$

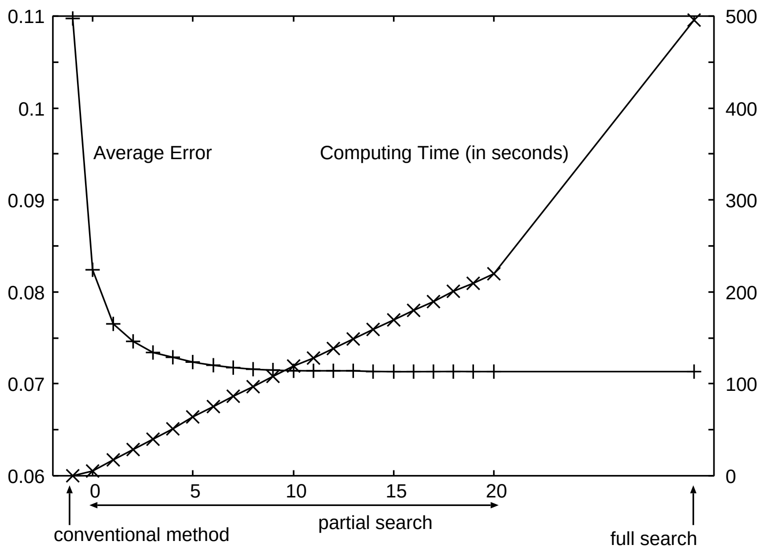
where $C' = C_{v-s} \cup C_{v-s+1} \cup \cdots \cup C_{v+s}$ for some appropriate fixed positive integer s, and v is an integer such that

$$\frac{v-\frac{1}{2}}{k^2} \leq I(b'_{i',j'}) < \frac{v+\frac{1}{2}}{k^2}.$$

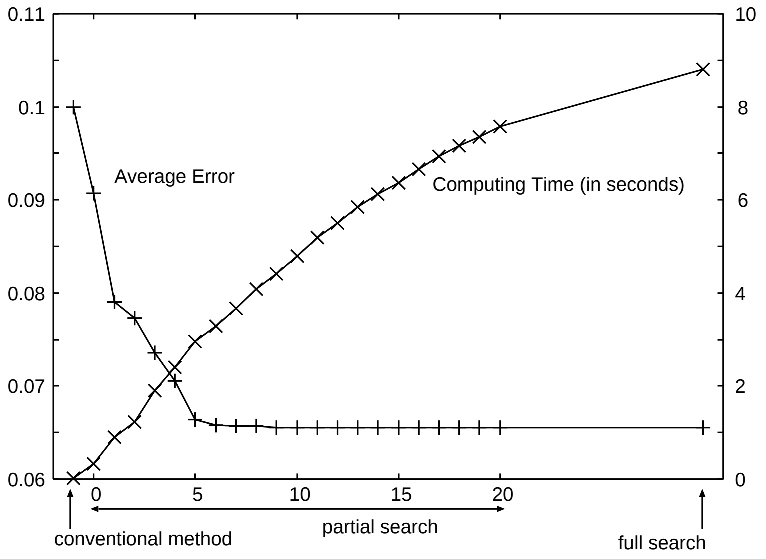
Note that $I(b'_{i',j'}) \approx \frac{v}{k^2}$ and thus C' includes characters with the intensity level close to $I(b'_{i',j'})$. We call such value s *the search range* of the local exhaustive search.

We have determined the appropriate value of the search range for the local exhaustive search by experiments using the CPU. Let the average error be $\frac{\mathrm{Error}(A,R)}{n^2}$ for the error $\mathrm{Error}(A,R)$ defined in (5). Figure 8 shows the average error and the computing time for generating ASCII/JIS art images of Lena gray-scale image in Figure 2 of size 1024×1024 using the conventional method, the local exhaustive search with search range from 0 to 20, and the local exhaustive search with full search (i.e. with unlimited search range). Clearly, the local exhaustive search with larger search range takes a longer time but smaller average error. In particular, the computing time is almost proportional to the search range. Thus, we have the trade-off between the computing time and the average error. Since the average error with search range 5 is almost the same as the local exhaustive search for full range, we have determined that the local exhaustive with search range 5 is good enough to obtain the same quality ASCII/JIS art images as the full search. From the graphs in Figure 8, the local exhaustive search with search range 5 is 7.7 times and 3.0 times faster than the full search for JIS Kanji code characters and ASCII code characters, respectively.

Step 3 just computes a bitmap image $B = (b_{i,j})$ by formula (4) from the ASCII/JIS art $B' = (b'_{i',j'})$. This can be done in an obvious way.



(1) JIS Kanji code characters



(2) ASCII Code characters

Figure 8. The average error and the computing time of ASCII/JIS art generation

5. GPU Implementation

Our GPU implementation of the local exhaustive search for generating an ASCII/JIS art is shown in this section.

We briefly explain CUDA architecture that we will use. NVIDIA provides a parallel computing architecture called *CUDA* on NVIDIA GPUs. CUDA uses two types of memories in the NVIDIA GPUs: *the global memory* and *the shared memory* [20]. The global memory is implemented as an off-chip DRAM of the GPU, and has large capacity, say, 1.5-6 Gbytes, but its access latency is very long. The shared memory is an extremely fast on-chip memory with lower capacity, say, 16-48 Kbytes. Figure 9 illustrates the CUDA hardware architecture.

CUDA parallel programming model has a hierarchy of thread groups called *grid*, *block* and *thread*. A single grid is organized by multiple blocks, each of which has equal number of threads. The blocks are allocated to streaming multiprocessors such that all threads in a block are executed by the same streaming multiprocessor

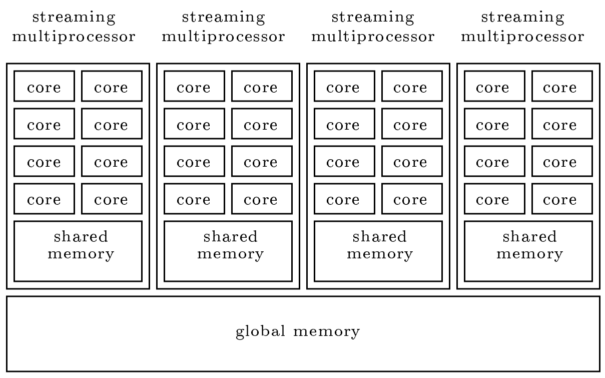


Figure 9. CUDA hardware architecture

in parallel. All threads can access to the global memory. However, threads in a block can access to the shared memory of the streaming multiprocessor to which the block is allocated. Since blocks are arranged to multiple streaming multiprocessors, threads in different blocks cannot share data in the shared memories.

We are now in a position to explain how we implement three steps of our ASCII/JIS art generation using the local exhaustive search. We assume that the adjusted image of an original image A is stored in the global memory in advance, and the implementation writes the resulting ASCII/JIS art image B' in the global memory. Further, we assume that the bitmap image of all characters in C and the blurred image of every character are also stored in the global memory.

To implement Step 1, $\frac{n^2}{k^2}$ CUDA blocks are invoked one for each block $A_{i',j'}$ of an image A. Let $B(i',j')$ $(0 \leq i',j' \leq \frac{n}{k}-1)$ denote a CUDA block assigned to a block $A_{i',j'}$. Each CUDA block $B(i',j')$ is responsible for computing the error matrix $E = (e_{i,j})$ of the corresponding block using the shared memory. For this purpose, $B(i',j')$ copies pixel values in A of the affected region $\mathcal{A}_{i',j'}$ in the shared memory. After that, each CUDA block $B(i',j')$ computes the average intensity level $I(A_{i',j'})$ by computing formula (7), and selects a character c in C_u satisfying formula (6). Finally, the error matrix $E = (e_{i,j})$ of the corresponding block is computed from the blurred image of c and pixel values in A of the affected region $\mathcal{A}_{i',j'}$. The error matrix E of the resulting block is copied to the global memory.

In Step 2, the local exhaustive search to evaluate formula (9) is performed in parallel using multiple CUDA blocks. However, the local exhaustive search for adjacent blocks cannot be executed in parallel, because the application of the Gaussian filter to adjacent blocks affects each other. Thus, we partition blocks into four groups such that
Group 1: even columns and even rows,
Group 2: odd columns and even rows,
Group 3: even columns and odd rows, and
Group 4: odd columns and odd rows.
The reader should refer to Figure 10 illustrating the groups. We use $\frac{4n^2}{k^2}$ CUDA blocks, and perform the local exhaustive search in all blocks of each group. Note that, if $k \geq 2w$ then the Gaussian filter of two blocks in a group never affect each other, where the bitmap image of a character is $k \times k$ and the size of the Gaussian filter is $(2w+1) \times (2w+1)$. In other words, the affected regions illustrated in Figure 6 of a particular group do not overlap each other. Actually, in our experiment, we choose $k = 16$ and $w = 3$. Step 2 performs the local exhaustive search for Group 1, Group 2, Group 3, and Group 4, in turn. A CUDA block is invoked for each block of a group. The CUDA block copies the error matrix corresponding to

the affected region in the global memory to the shared memory. After that, each CUDA block evaluates the right-hand side of formula (9) to find the replacement character. Finally, the error matrix $E = (e_{i,j})$ of the corresponding block is computed and the error matrix E of the resulting block is copied to the global memory in the same way as Step 1.

To implement Step 3, one CUDA block is used to generate a block of the bitmap image $B = (b_{i,j})$ by formula (4) from the ASCII/JIS art $B' = (b'_{i',j'})$. This can be done in an obvious way.

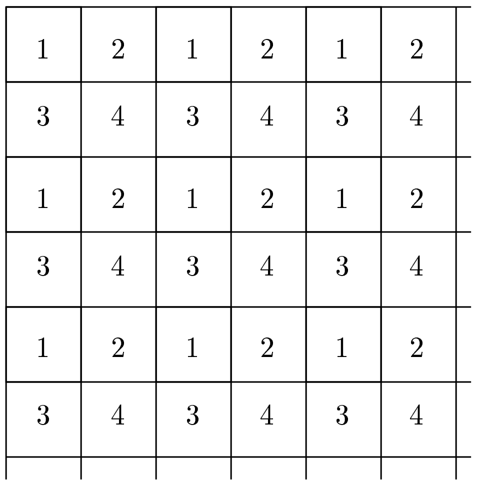


Figure 10. Groups of blocks

Table 1. Computing time (in seconds) for generating ASCII/JIS art images by the conventional method

JIS art generation			
Image size	256×256	512×512	1024×1024
Intel CPU	0.8619×10^{-3}	3.441×10^{-3}	14.14×10^{-3}
NVIDIA GPU	1.457×10^{-3}	1.844×10^{-3}	2.391×10^{-3}
Speed-up	0.5915	1.866	5.913
ASCII art generation			
Image size	256×256	512×512	1024×1024
Intel CPU	0.8989×10^{-3}	3.560×10^{-3}	14.21×10^{-3}
NVIDIA GPU	1.562×10^{-3}	1.969×10^{-3}	2.615×10^{-3}
Speed-up	0.5754	1.808	5.434

6. Experimental results

In this section, we will show the resulting images and the computing time.

We have used Lena gray-scale images in Figure 2 of size 256×256, 512×512, and 1024×1024. We use a set of 7310 characters in the JIS Kanji code with 16×16 pixels and a set of 95 characters in the ASCII code with 16×8 pixels. A Gaussian filter of size 7×7 with parameter $\sigma = 2.0$ is used. Figure 11 shows the resulting ASCII art images generated using JIS Kanji code characters from Lena gray-scale image of size 1024×1024. Also, Figure 12 shows the resulting ASCII/JIS art images generated using ASCII code characters. As we have shown in Section 4,

Table 2. Computing time (in seconds) for generating ASCII/JIS art images by our method using the LES with search range 5

JIS art generation			
Image size	256×256	512×512	1024×1024
Intel CPU	4.009	16.10	64.03
NVIDIA GPU	94.69×10^{-3}	226.1×10^{-3}	714.9×10^{-3}
Speed-up	42.33	71.20	89.56
ASCII art generation			
Image size	256×256	512×512	1024×1024
Intel CPU	0.1701	0.7179	2.952
NVIDIA GPU	7.984×10^{-3}	17.66×10^{-3}	51.70×10^{-3}
Speed-up	21.30	40.65	57.09

Table 3. Data transfer time (in seconds) between a Host PC to a GPU

Image size	256×256	512×512	1024×1024
Host PC$\leftarrow$GPU	0.08291×10^{-3}	0.2069×10^{-3}	0.7143×10^{-3}
Host PC$\rightarrow$ GPU	0.08729×10^{-3}	0.2515×10^{-3}	0.8073×10^{-3}

the search range of the local exhaustive search should be 5 to balance the running time and the quality of resulting images. Hence, we have executed the conventional method and our method using the local exhaustive search with search range 5. The resulting ASCII/JIS art images by our method can reproduce the details and the tones of the original Lena image, and the quality is much better than those by the conventional method. In particular, the edges of images are sharper.Figure 13 shows the blurred images of ASCII/JIS art images shown in Figures 11 and 12, which are obtained by the same parameter $\sigma = 2.0$. We can think that these images are projected in human retina. The readers should have no difficulty to confirm that blurred images obtained by our method are very similar to the original gray scale image. Also, we can verify the goodness of our method by evaluating the SNR of the blurred images. The SNR of the blurred images by the conventional method for the image of size 1024×1024 are 15.3dB and 17.8dB for JIS and ASCII art images, respectively, while those by our method is 19.0dB and 21.3dB, respectively.

We have evaluated the computing time for generating the ASCII/JIS art images. We have used a Intel PC using Xeon E5-2430 running in 2.2GHz to evaluate the implementation by sequential algorithms. We also used NVIDIA GeForce GTX 780 Ti, which has 2880 processing cores in 15 SMX units [21]. Table 1 shows the computing time for generating the ASCII/JIS art images by the conventional method. Also, Table 2 shows the computing time for generating the ASCII/JIS art images by our method using the local exhaustive search with search range 5. Our method using the local exhaustive search takes much more time than the conventional method. However, by using the GPU, the computing time can be reduced by a factor of 21.30-89.56. Our method takes 2.952s for the Lena image of size 1024×1024 using the ASCII code. The computing time can be reduced to 51.7ms using the GPU. Even if the JIS Kanji code is used, the computing time is 0.7149s by the GPU acceleration. This computing time is acceptable for most applications of amusement purpose. Table 3 shows the data transfer time between a Host PC and the global memory of the GPU. We can confirm that the data transfer time is negligible, because the computation of ASCII/JIS art generation is costly.

7. Conclusions

The main contribution of this paper is to propose a new technique to generate an ASCII art image that reproduces the original tone and the details of input gray-scale images. We have presented a new technique using the local exhaustive search to optimize binary images for printing based on the characteristic of the human visual system. The resulting ASCII art images by our new method can reproduce the details and the tones of original gray-scale images. To accelerate ASCII art generation by our method, we have implemented it in the GPU. The experimental results show that the GPU implementation can achieve a speedup factor up to 89.56 over the conventional CPU implementation.

References

[1] L. Carroll, *Alice's Adventures in Wonderland*, Macmillan, 1865.
[2] X. Xu, L. Zhang, and T.T. Wong, *Structure-based ASCII art*, ACM Transactions on Graphics (SIGGRAPH 2010 issue) 29 (2010), pp. 52:1–52:9.
[3] D.L. Lau and G.R. Arce, *Modern Digital Halftoning*, Marcel Dekker, 2001.
[4] D. Knuth, *Digital halftones by dot diffusion*, ACM Trans. Graphics 6-4 (1987), pp. 245–273.
[5] Y. Ito and K. Nakano, *FM screening by the local exhaustive search with hardware acceleration*, International Journal on Foundations of Computer Science 16 (2005), pp. 89–104.
[6] Y. Ito and K. Nakano, *A new FM screening method to generate cluster-dot binary images using the local exhaustive search with FPGA acceleration*, International Journal on Foundations of Computer Science 19 (2008), pp. 1373–1386.
[7] M. Analoui and J. Allebach, *Model-based halftoning by direct binary search*, in *Proc. SPIE/IS&T Symposium on Electronic Imaging Science and Technology*, Vol. 1666, SPIE, 1992, pp. 96–108.
[8] IROMSOFT, *Text artist* URL `http://www.hm.h555.net/irom/`.
[9] Y. Furuta, J. Mitani, and Y. Fukui, *A method for generating ascii-art images from a character sequence by adjusting the kerning*, Tech. rep., IPSJ, 2010.
[10] P.D. O'Grady and S.T. Rickard, *Automatic ASCII art conversion of binary images using non-negative constraints*, in *Proc. of the Irish Signal and Systems Conference*, Institute of Engineering and Technology, 2008, pp. 186–191.
[11] W.W. Hwu, *GPU Computing Gems Emerald Edition*, Morgan Kaufmann, 2011.
[12] D. Man, K. Uda, H. Ueyama, Y. Ito, and K. Nakano, *Implementations of a parallel algorithm for computing Euclidean distance map in multicore processors and GPUs*, International Journal of Networking and Computing 1 (2011), pp. 260–276.
[13] K. Ogawa, Y. Ito, and K. Nakano, *Efficient Canny Edge Detection Using a GPU*, in *Proc. of International Conference on Networking and Computing*, Nov., IEEE CS Press, 2010, pp. 279–280.
[14] A. Uchida, Y. Ito, and K. Nakano, *Fast and Accurate Template Matching using Pixel Rearrangement on the GPU*, in *Proc. of International Conference on Networking and Computing*, Dec., IEEE CS Press, 2011, pp. 153–159.
[15] Y. Ito, K. Ogawa, and K. Nakano, *Fast Ellipse Detection Algorithm using Hough Transform on the GPU*, in *Proc. of International Conference on Networking and Computing*, Dec., IEEE CS Press, 2011, pp. 313–319.
[16] A. Uchida, Y. Ito, and K. Nakano, *An Efficient GPU Implementation of Ant*

Colony Optimization for the Traveling Salesman Problem, in *Proc. of International Conference on Networking and Computing*, Dec., IEEE CS Press, 2012, pp. 94–102.

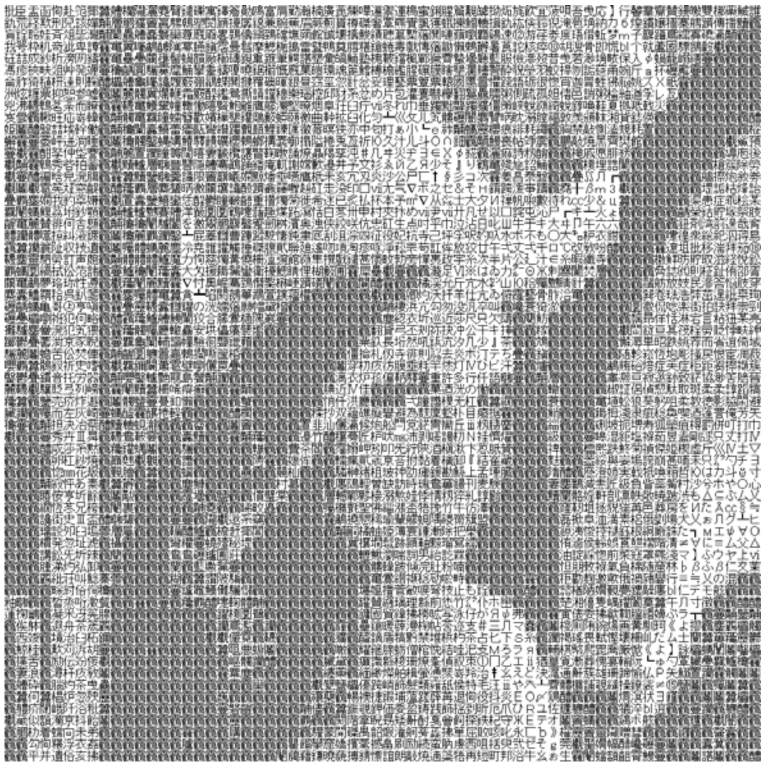
[17] A. Kasagi, K. Nakano, and Y. Ito, *Offline permutation algorithms on the discrete memory machine with performance evaluation on the GPU*, IEICE Transactions on Information and Systems Vol. E96-D (2013), pp. 2617–2625.

[18] A. Kasagi, K. Nakano, and Y. Ito, *An Optimal Offline Permutation Algorithm on the Hierarchical Memory Machine, with the GPU implementation*, in *Proc. of International Conference on Parallel Processing*, Oct., IEEE CS Press, 2013, pp. 1–10.

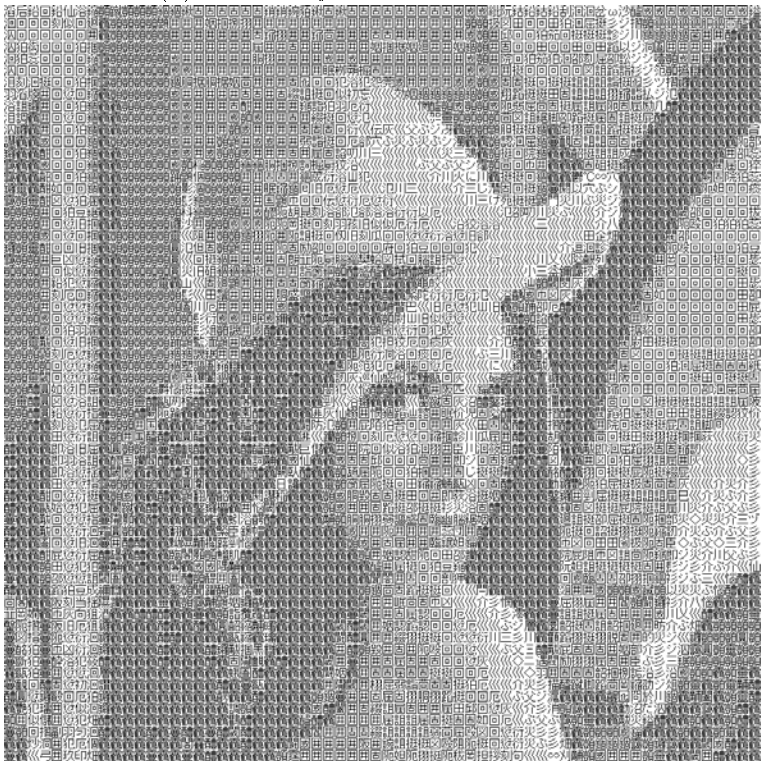
[19] D. Man, K. Nakano, and Y. Ito, *The Approximate String Matching on the Hierarchical Memory Machine, with Performance Evaluation*, in *Proc. of International Symposium on Embedded Multicore/Many-core System-on-Chip*, Sept., IEEE CS Press, 2013, pp. 79–84.

[20] NVIDIA Corporation, *NVIDIA CUDA C programming guide version 5.0* (2012).

[21] NVIDIA Corporation, *NVIDIA next generation CUDA compute architecture: Kepler GK110 whitepaper* (2013).

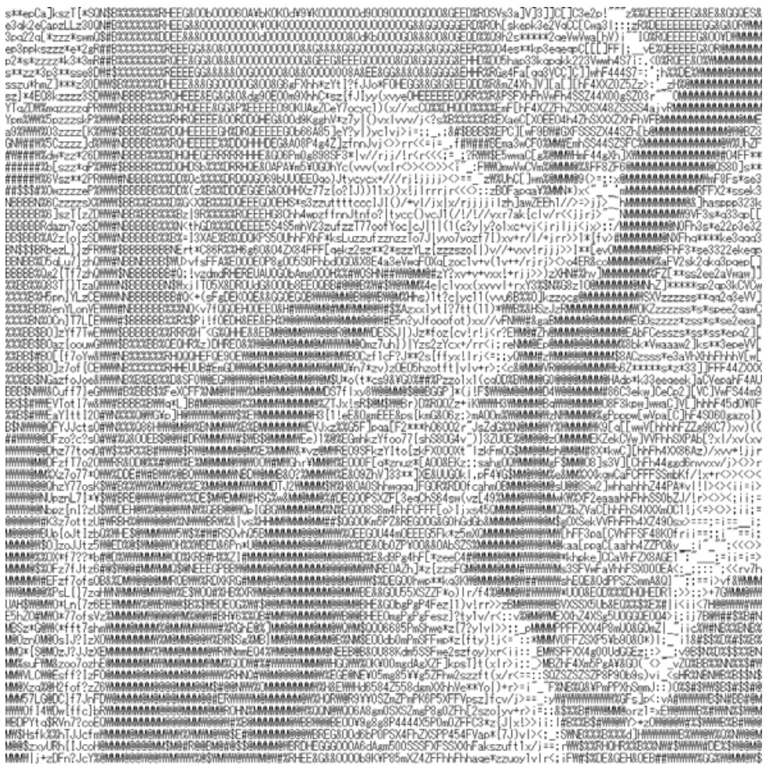


(1) JIS art by conventional method

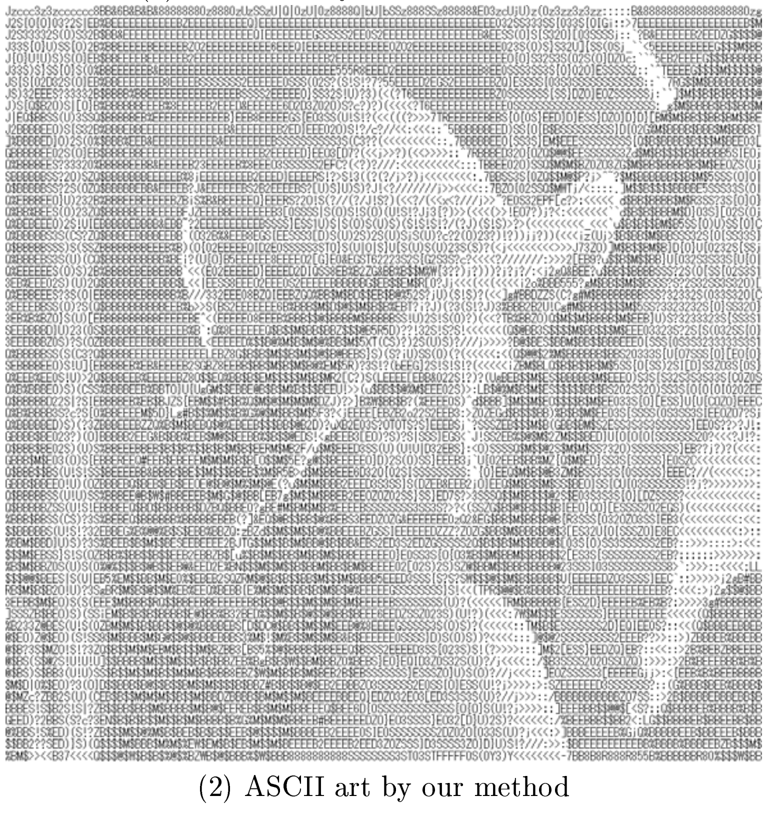


(2) JIS art by our method

Figure 11. The resulting JIS art images using JIS Kanji code characters

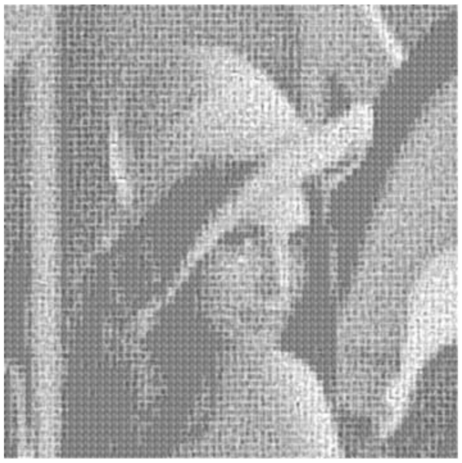


(1) ASCII art by conventional method

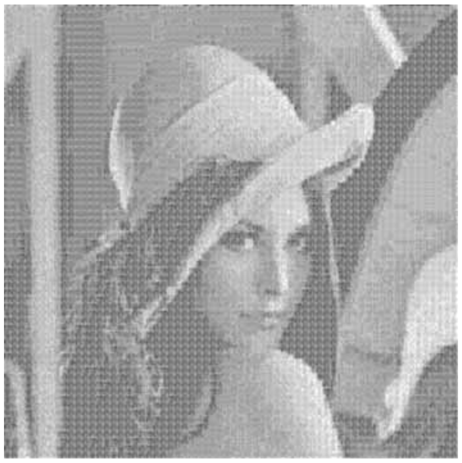


(2) ASCII art by our method

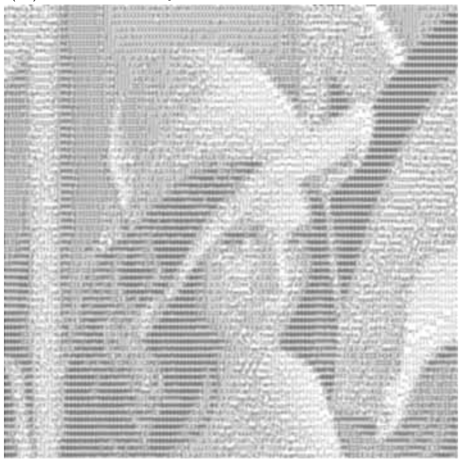
Figure 12. The resulting ASCII art images using ASCII code characters



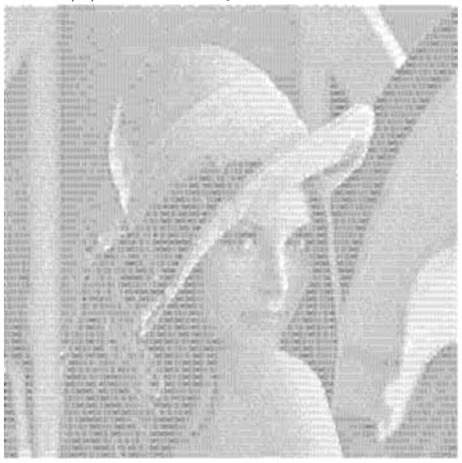
(1) JIS art by conventional method



(2) JIS art by our method



(3) ASCII art by conventional method



(4) ASCII art by our method

Figure 13. Blurred images of ASCII/JIS art images