Development of a Method for Extracting and Recognizing Graph Elements in Mathematical Graphs for Automating Translation of Tactile Graphics

Jianjun CHEN * · Noboru TAKAGI *

Tactile graphics are images that use raised surfaces so that a visually impaired person can feel them. Tactile graphics are necessary for visually impaired students when they study mathematics and science. Since producing tactile graphics is not simple task, an intelligent computer-aided system for assisting the production of tactile graphics is needed. Mathematical graph recognition from printed materials plays an important role in developing such a system. So, this paper focuses on part of a method of mathematical graph recognition. A mathematical graph consists of character strings, mathematical formulas and graph elements such as the rays representing the $x$-axis and the $y$-axis, and the straight lines or the curves representing functions or equations. Graph elements are drawn not only by solid lines, but also broken lines. This paper discusses a method for extracting and recognizing graph elements from mathematical graphs. The effectiveness of our method is evaluated by computer experiments.

Keywords : Pattern Recognition, Computer-aided Systems for People with Visual Impairments, Tactile Graphics, Scalable Vector Graphics, Spline Interpolation, Bézier Curves

1. Introduction

Graphs are frequently used to present functions and equations in mathematics and science. Since most of these graphs are in visual form, they cannot be utilized by visually impaired users. Through tactile graphics, pictures can be understood by the visually impaired. This is because tactile graphics are produced by raised patterns which can be felt with the fingertips. Although tactile graphics are useful for visually impaired students when they learn mathematics and science, in 80% of Japanese schools for the visually impaired, there is no department to produce tactile teaching materials [1]. Usually, teachers produce most tactile graphics using less intelligent computer-aided systems. So, a better computer-aided system of making tactile graphics is needed.

InfyProject [2] is a volunteer organization for helping people with visual impairments in scientific fields. One of the purposes of InfyProject is to digitize scientific documents such as mathematics journals and books, and it provides their tactile materials to the visually impaired. To give the service above, InfyProject developed a software system called InfyReader which can translate printed mathematical expressions into digital expressions such as LaTeX and MathML. InfyReader contributes to the efficient production of tactile materials. However, because there is no intelligent system to translate printed mathematical graphs into SVG (Scalable Vector Graphics) [3] images, the efficiency of works related to the translations is disturbed. Note that the DAISY (Digital Accessible Information System) consortium [4], which develops, maintains, and promotes the international standard of digital books for people with visual impairments, recently adopted SVG as the standard for digital figures.

The work, reported by Kruka et al. [5], is done on automating tactile graphics production from scalable vector graphics. However, many images are not yet in SVG format. The Tactile Graphics Assistant (TGA) is a software to assist the translation of bitmap images to tactile representations [6]. All of the steps of the workflow of TGA are in a batch, but between each step human intervention and validation is necessary [7]. It is our hope to automate the translations of figures in mathematics and science books to tactile representations.
To develop a system to translate mathematical graphs into tactile graphics automatically, mathematical graph recognition techniques are needed. So far, many graph recognition methods have been developed. Fuda et al. [8] studied a graph recognition method requiring that graphs must satisfy many assumptions; a graph, for example, has to be drawn inside a rectangular area that is specified by the $x$-axis and the $y$-axis. The graph recognition methods introduced by the literatures [9, 10], and [11] must also satisfy assumptions about graphs. However, many mathematical graphs do not satisfy all of the aforementioned assumptions.

To facilitate the production of tactile graphics, we are developing a computer-aided system [12] for automating translation of printed mathematical graphs into SVG images. The following descriptions summarize the characteristics of the mathematical graphs we focus on.

1. Characters and mathematical formulas may be distributed in and around the graph.

2. A character string or a mathematical formula may not lie on the correct orientation (i.e., on the horizontal orientation).

3. Graphs may contain several types of broken lines. A mathematical graph includes the following three elements: (1) solid line graph elements, (2) broken line graph elements, and (3) elements from character strings and mathematical formulas. This paper focuses on mathematical graph recognition.

This paper is organized as follows. Section 2 is about the separation of mathematical graphs into three regions: solid line elements, character elements, and broken line elements. Section 3 shows the method of extraction of broken line graph elements. Section 4 introduces the method for extraction of solid line graph elements. Section 5 presents the techniques for fitting and classifying graph elements. Experimental results and discussion are described in Section 6, and this paper is concluded in Section 7, where future works are also postulated.
2. Separation of Mathematical Graphs [12]

We have developed a method to separate a bitmap image of a mathematical graph into three parts: solid line elements, broken line elements, and character elements. Fig.1 shows the outline of the separation method. A clustering method is applied to the set of small elements, and as the result finds clusters so that each of them includes only broken line elements. The characteristics of these clusters are shown below.

1. For each of the clusters, G, almost all elements in G are elements of the same broken line. The elements of a single broken line are often divided into different clusters.
2. So, the problem is how to find a cluster which includes all elements of a single broken line. We will discuss this issue in the following sections.

3. Extraction of Broken Line Graph Elements

Fig.2 shows examples of broken lines which we focus on. There are dotted lines and chain lines. Chain lines are further broken down into types 1, 2, and 3. In this section, we describe methods that classify a cluster from the previous section into a dotted line or a chain line.

A dotted line consists of homogeneous rectangular elements, while a chain line is composed by two different kinds of rectangular elements: short elements and long elements. Therefore, elements of a dotted line can be classified into only one group, homogeneous rectangular elements. Similarly, elements of a chain line are classified into two different groups: one is for short elements, and the other is for long elements. If we can evaluate the number of homogeneous groups in a cluster, it enables us to classify the cluster into a dotted line or a chain line. To evaluate the optimal number of homogeneous groups (i.e., clusters), we measure by two cluster validities, $V_{DB}(\cdot)$ and $V_{D}(\cdot)$ [13], whose definitions are given below.

Given a set of $k (\geq 2)$ clusters, $\Gamma = \{G_1, ..., G_k\}$,

$$V_{DB}(\Gamma) = \frac{1}{k} \sum_{i=1}^{k} \max_{j \neq i} \left( \alpha_i + \alpha_j \right) \frac{\alpha_i}{\alpha_i + \alpha_j}$$

where for $i = 1, ..., k$, $\alpha_i = \sum_{x \in G_i} \frac{x - \bar{v}_i}{|G_i|}$ and

$$V_D(\Gamma) = \min_{1 \leq i \leq k} \left( \min_{1 \leq j \leq k, j \neq i} \left( \max_{x \in S} \delta(x, y) \right) \right)$$

Here, for any clusters $S$ and $T$, $\Delta(S) = \max_{x \in S, y \in T} \delta(x, y)$ and $\hat{\Delta}(S, T) = \min_{x \in S, y \in T} \delta(x, y)$, and $\delta(x, y)$ is the distance between $x$ and $y$.

Note that the larger the value of $V_{DB}(\cdot)$, the better the clustering result. Similarly, the smaller the value of $V_{D}(\cdot)$, the better the clustering result. These two cluster validities are not defined when $k = 1$, therefore, we introduce a fuzzy inference system to avoid this disadvantage.

3.1 Dotted Line Classification

In this section, we describe the dotted line classification method. The following description is the procedure for the dotted line classification method.

**Input**: A cluster, G, of broken line elements.

**Output**: If $G$ is a dotted line, return Yes, otherwise return No.

**Step 1**: Single-linkage clustering is applied to cluster $G$, and let $\Gamma_i$ be the result when the number of clusters is $k (k = 1, 2, 3, 4, 5)$. Calculate $V_{DB}(\Gamma_i)$ and $V_D(\Gamma_i)$ for every $k$.

**Step 2**: Apply a fuzzy inference system to $V_{DB}(\Gamma_k)$ and $V_D(\Gamma_k)$, if $G$ is classified as a dotted line by the fuzzy inference system, then return Yes, otherwise return No.

In the single-linkage clustering of Step 1, every element, $e$, in $G$ is represented by two characteristics: the number of the pixels of $e$, and the length of the long side of $e$. The calculation scheme of the fuzzy inference system is based on Mamd[14], but the minimum
operator is exchanged with the product operator. The fuzzy inference system has four arguments, \( x_1, x_2, x_3, \) and \( x_4 \), which are defined as follows. Let \( G = \{ e_1, \ldots, e_\ell \} \), a set of broken line elements, and let \( \Gamma_k \) be a set of clusters of set \( G \); and then
\[
\begin{align*}
x_1 &= \max_{e \in \Gamma_k} \left( \sum_{i=1}^{j} w_i(x_i) \right), & x_2 &= \min_{e \in \Gamma_k} \left( \sum_{i=1}^{j} w_i(x_i) \right),
\end{align*}
\]
where \( p(e) \) is the number of the pixels of element \( e \) and \( \ell(e) \) is the length of the long side of element \( e \).

Fuzzy if–then rules are given below, and the membership functions are shown in Fig.3.

**Rule 1:** If \( x_1 \) is large, \( x_3 \) is large, and \( x_4 \) is large, then \( G \) is a dotted line.

**Rule 2:** If \( x_2 \) is small, \( x_3 \) is large, and \( x_4 \) is large, then \( G \) is a dotted line.

**Rule 3:** If \( x_1 \) is small, then \( G \) is not a dotted line.

**Rule 4:** If \( x_2 \) is large, then \( G \) is not a dotted line.

**Rule 5:** If \( x_3 \) is small, then \( G \) is not a dotted line.

**Rule 6:** If \( x_4 \) is small, then \( G \) is not a dotted line.

A cluster is classified as a dotted line if the output value of the fuzzy inference systems is more than or equal to 0.5.

### 3.2 Chain Line Classification

Next, we will discuss a chain line classification method, which distinguishes clusters of elements of chain lines. If a cluster is classified as a chain line, the chain line classification method also gives its type. The following description is the procedure.

**Input:** A cluster, \( G \), of broken line elements.

**Output:** If \( G \) is classified as a chain line, return the type of \( G \), otherwise return No.

**Step 1:** Apply single–linkage clustering to \( G \) by setting the number of clusters to 2, and then divide \( G \) into two groups. Assign label ‘a’ to elements of one group, and label ‘b’ to elements of the other group. We then have a sequence of labels for \( G \).

**Step 2:** Calculate similarity, \( S_p(G) \) \((p = 1, 2, 3)\), between the sequence and the template of a type \( p \) chain line. Here, the template of type 1 chain lines is \( ababab\cdots \). Similarly, those of types 2 and 3 are \( abababab\cdots \) and \( abbabab\cdots \), respectively.

**Step 3:** If similarity \( S_p(G) \) is equal to the number of elements of \( G \), then classify \( G \) as a type \( p \) chain line and return \( p \), otherwise return No.

For the sequence obtained by Step 1 and the template of a type \( p \) chain line, similarity \( S_p(G) \) is defined as the largest number of successively matching labels.

### 3.3 Broken Line Classification

Lastly, the procedure for the broken line classification method is described below.

**Input:** A set of clusters, \( \Gamma = \{ G_1, \ldots, G_l \} \), from Section 2.

**Output:** A set of dotted line clusters, \( \Delta \), sets of type \( p \) chain line clusters, \( X_p \) \((p = 1, 2, 3)\), and a set of clusters, \( \Phi \).

**Step 1:** Set \( \Delta \leftarrow \phi \), \( X_p \leftarrow \phi \) \((p = 1, 2, 3)\), and \( \Phi \leftarrow \phi \).

**Step 2:** If \( \Gamma \) is empty, output \( \Delta \), \( X_p \) \((p = 1, 2, 3)\), and \( \Phi \), and stop the procedure.

**Step 3:** Select \( G_i \) from \( \Gamma \), and set \( \Gamma \leftarrow \Gamma - \{ G_i \} \). If \( G_i \) includes only one element, set \( \Phi \leftarrow \Phi \cup \{ G_i \} \) and go to Step 2.

**Step 4:** Apply the dotted line classification method to \( G_i \). If \( G_i \) is classified as a dotted line, set \( \Delta \leftarrow \Delta \cup \{ G_i \} \) and go to Step 2.

**Step 5:** Apply the chain line classification method to \( G_i \). If \( G_i \) is classified as a type \( p \) chain line, set \( X_p \leftarrow X_p \cup \{ G_i \} \) and go to Step 2.

**Step 6:** Divide set \( G_i \) into three groups, \( G_i', G_i^-, \) and \( G_i^0 \), in the following way. Suppose \( S_p(G) \) is the greatest among the three similarities,
$S_i(G_i), S_2(G_i),$ and $S_3(G_i)$. Then, separate the elements of the sequence for $G_i$ which gives similarity $S_i(G_i)$, and let $G_i'$ denote the set of these elements. Furthermore, let $G_i''$ and $G_i'''$ denote the set of elements in $G_i$ which are located on the left and the right side of $G_i'$, respectively.

**Step 7:** $G_i'$ is classified as a type $p$ chain line. Add $G_i'$ to set $X_p$, and also add $G_i''$ and $G_i'''$ to set $\Gamma$. Go to Step 2.

### 3.4 Merging Clusters

Since the broken line classification method divides a cluster into several groups until every cluster is classified into one of the four types of broken lines, we need a merging process that combines clusters consisting of elements from the same broken line into a single cluster. If two clusters, $G_1$ and $G_2$, satisfy the following two geometric characteristics, it is plausible that $G_1$ and $G_2$ are merged into a single cluster: (1) the gradient for the broken lines corresponding to $G_1$ is almost equal to the gradient for the broken line corresponding to $G_2$, and the two broken lines are located closely to each other, and (2) broken lines $G_1$ and $G_2$ are separated by some obstacles such as character strings. So our merging process is based on the following two criteria:

1. Similar geometric description
2. Existence of interference

Our merging process thus consists of two different merging processes.

#### 3.4.1 Disposition Criterion

The following description explains the flow of the merging process based on the first criterion.

**Input:** A set, $\Gamma$, of clusters from Section 3.3.

**Output:** A set of clusters after merging.

**Step 1:** Select a pair of clusters, $(G_i, G_j)$, from $\Gamma$ which is not tested yet. If there exists no such pair, output $\Gamma$ and stop this procedure.

**Step 2:** If the types of broken lines $G_i$ and $G_j$ are different, go to Step 1.

**Step 3:** Apply pair $(G_i, G_j)$ to a fuzzy inference system. If the result from the fuzzy inference system is positive (i.e., $G_i$ and $G_j$ are part of the same broken line), then merge $G_i$ and $G_j$ into a single cluster and go to the next step, otherwise go to Step 1.

**Step 4:** Let $G$ be the cluster from the previous step. Update $\Gamma$ by setting $\Gamma \leftarrow (\Gamma-\{G_i, G_j\}) \cup \{G\}$, and go to Step 1.

Let $G_1$ and $G_2$ be clusters corresponding to dotted lines. If these two clusters satisfy the following four geometric characteristics, then it is plausible that these two clusters are part of the same broken line.

1. The number of pixels of any element in $G_1$ is almost equal to the number of pixels of any element in $G_2$.
2. The length of the long side of any element in $G_1$ is almost equal to the length of the long side of any element in $G_2$.
3. The distance between $G_1$ and $G_2$ is short.
4. The curvature at the intersection determined by lengthening dotted lines $G_1$ and $G_2$ is small.

The curvature for a digital curve at a point is calculated by the method introduced by literature [15]; that is, the curvature, $\kappa$, at point $(x_i, y_i)$ in Fig. 4(b) is defined as $\kappa = \left| \varphi_i - \varphi_{i-1} \right|$. Fuzzy inference is applied to evaluate degree, which represents how true it is that two clusters $G_1$ and $G_2$ are part of the same broken line. The fuzzy inference system has four arguments, $x_1, x_2, x_3,$ and $x_4$; and they have the following measurements concerning the four geometric characteristics for dotted lines.

1. $x_1 = \left| \frac{1}{|G_i|} \sum_{e \in G_i} p(e) - \frac{1}{|G_j|} \sum_{e \in G_j} p(e) \right|$, \hspace{1cm} (3)

where $p(e)$ is the number of the pixels of element $e$.

2. $x_2 = \left| \frac{1}{|G_i|} \sum_{e \in G_i} \ell(e) - \frac{1}{|G_j|} \sum_{e \in G_j} \ell(e) \right|$, \hspace{1cm} (4)

where $\ell(e)$ is the length of the long side of element $e$.

3. $x_3$ is the shortest distance between $G_1$ and $G_2$; the shortest distance is determined by two endpoints of $G_1$ and $G_2$ (see Fig. 4(a)).

4. $x_4$ is the curvature at the intersection determined by lengthening dotted lines $G_1$ and $G_2$ (see Fig. 4(b)).

The fuzzy if-then rules of the fuzzy inference system is shown below; in the following rules, $G_1$ and $G_2$ stand for clusters of elements from dotted lines.

**Rule 1:** If $x_1$ is small, $x_2$ is small, $x_3$ is short, and $x_4$ is small, then $G_1$ and $G_2$ are part of the same dotted
cubic spline function, \( y = s(x) \), is calculated using \( G_1 \) and \( G_2 \); the spline function is derived from the knots, which are endpoints of the elements in \( G_1 \) and \( G_2 \). We then count the number of pixels, \((x, y)\), satisfying the following conditions.

1. \((x, y)\) satisfies condition \( y = s(x) \), and is located between \( G_1 \) and \( G_2 \).

2. \((x, y)\) is a black pixel in the original image. \( G_1 \) and \( G_2 \) are finally determined to be part of the same broken line if this number is large enough, and then merged into a single cluster. In this method, if the number of the pixels exceeds half of the length of the part of \( g \), which is the graph of \( y = s(x) \), located between \( G_1 \) and \( G_2 \), then the two clusters are merged into a single cluster.

Let \( G_1 \) and \( G_2 \) be two clusters of elements of dotted lines. Fuzzy if–then rules of the fuzzy inference system that evaluates similarities are then shown below.

**Rule 1**: If \( x_1 \) is small and \( x_2 \) is small, then \( G_1 \) and \( G_2 \) are part of the same dotted line.

**Rule 2**: If \( x_1 \) is large, then \( G_1 \) and \( G_2 \) are not part of the same dotted line.

**Rule 3**: If \( x_2 \) is large, then \( G_1 \) and \( G_2 \) are not part of the same dotted line.

In these fuzzy if–then rules, the value of \( x_1 \) and \( x_2 \) are calculated by equations (3) and (4), respectively. Definitions of membership functions of the two fuzzy sets are omitted due to the limitations of space. Note that if \( G_1 \) and \( G_2 \) are chain lines of the same type, then two variables regarding the average of lengths are needed: one is for short segments and the other is for long segments. Similarly, we need two variables regarding the average of lengths. Therefore, when \( G_1 \) and \( G_2 \) are chain lines of the same type, the fuzzy inference systems has 5 fuzzy if–then rules.

### 3.4.2 Interference Criterion

Almost all of the broken line elements are in rectangular shapes; however, some of them are not. Non-rectangular elements are not classified as broken line elements. Furthermore, a broken line element overlapped with a graph element is not classified as a broken line element either. For this reason, a single broken line is sometimes divided into two or more parts; and therefore we need to introduce a merging process for the second criterion. This merging process is based on spline interpolations.

First, we select a pair of clusters, \( G_1 \) and \( G_2 \), from the previous section. Then, \( G_1 \) and \( G_2 \) are examined by a fuzzy inference system whose output means a similarity between \( G_1 \) and \( G_2 \). Here, the higher the similarity between \( G_1 \) and \( G_2 \), the more plausible the fact that \( G_1 \) and \( G_2 \) are part of the same broken line. When the similarity between \( G_1 \) and \( G_2 \) is high, a natural

### 4. Extraction of Solid Line Graph Elements

From Section 2, large elements (i.e., solid line graph elements) are separated from small elements. A large element is often formed by overlapping two or more graph elements such as rays of the \( x \)-axis and the \( y \)-axis and so on. The method for extraction of solid line graph elements is based on two steps: segmentation and merging.
4.1 Segmentation of Large Elements

We introduce the following five procedures for splitting large elements: (1) thinning, (2) removing short branches, (3) finding intersections, (4) splitting, and (5) cutting hooklets.

(1) Thinning: We first apply a thinning procedure [16] to large elements, giving us skeletons of those elements.

(2) Removing Short Branches: A skeleton often includes many undesirable short branches, and therefore every branch whose length is less than a threshold value is removed from the skeleton.

(3) Finding Intersections: For every point, \( p \), in a skeleton, using \( 3 \times 3 \) spatial filters we identify whether \( p \) is an intersection point. The width of an original graph element is more than one pixel, and so not every intersection in the original image is a point. Therefore, an intersection includes more than one intersection point in the skeleton for a large element (See Fig.5). Thus, to group intersection points located in the same intersection, we apply single-linkage clustering to the set of intersection points.

(4) Splitting: By removing all intersection points in a skeleton, the skeleton is divided into small fragments. For a fragment, \( e \), if intersection points adjacent to the two endpoints of \( e \) are members of the same cluster, then we remove \( e \) from the set of fragments. Furthermore, a fragment is also removed if its size is less than a threshold value. The remaining fragments are called primitive elements.

(5) Cutting Hooklets: Due to the thinning algorithm, a fragment often has a hooklet at the extreme tip (See Fig.5(b)). To facilitate the merging process in the following section, we remove the hooklets of fragments. We use Wall and Danielsson [17] to check if a hooklet exists.

4.2 Merging Primitive Elements

We apply fuzzy inference to merge primitive elements, so that the merging result forms a graph element. In this section, we apply fuzzy inference twice. The first fuzzy inference system is applied to examine how geometrically similar two primitive elements are. The second fuzzy inference system is then applied to merge primitive elements selected by the first fuzzy inference system.

If two primitive elements, \( e_1 \) and \( e_2 \), satisfy the following three geometric characteristics, it is plausible that \( e_1 \) and \( e_2 \) are part of the same graph element.

(1) \( e_1 \) and \( e_2 \) have been connected at the same intersection before applying the segmentation process from Section 4.1.

(2) Let \( C \) be a curve which is part of a skeleton from Section 4.1, and suppose \( C \) includes \( e_1 \) and \( e_2 \). A curvature of \( C \) at a point \( p \) is then low, where \( p \) is the intersection point that divides \( e_1 \) and \( e_2 \).

(3) The width of the original graph element corresponding to \( e_1 \) is almost equal to the width of the original graph element corresponding to \( e_2 \).

The first fuzzy inference system has two arguments: curvature, \( x_1 \), and width, \( x_2 \). In the case where two primitive elements, \( e_1 \) and \( e_2 \), are connected to the same intersection, \( e_1 \) and \( e_2 \) are applied to the first fuzzy inference system. The fuzzy if-then rules of the first fuzzy inference system are shown below.

**Rule 1**: If \( x_1 \) is low and \( x_2 \) is small, then the similarity between the two primitive elements \( e_1 \) and \( e_2 \) is high.

**Rule 2**: If \( x_1 \) is high, then the similarity between the two primitive elements \( e_1 \) and \( e_2 \) is low.

**Rule 3**: If \( x_2 \) is large, then the similarity between the two primitive elements \( e_1 \) and \( e_2 \) is low.

The first fuzzy inference system checks the geometrical similarity between any pair of primitive elements. We then select primitive elements if the similarity exceeds a threshold value, and let \( E \) be the set of such primitive elements. Next, we apply the second fuzzy inference system to any pair of primitive elements in \( E \). The second fuzzy inference system has three arguments: \( x_1 \), \( x_2 \), and \( x_3 \); \( x_1 \) and \( x_2 \) are same as the two arguments of the first one, while the
third argument, $x_3$, takes the shortest distance between two primitive elements. The fuzzy if-then rules of the second fuzzy inference system are shown below.

**Rule 1:** If $x_1$ is low, $x_2$ is small, and $x_3$ is short, then the two primitive elements $e_1$ and $e_2$ are part of the same graph element.

**Rule 2:** If $x_1$ is high, then the two primitive elements $e_1$ and $e_2$ are not part of the same graph element.

**Rule 3:** If $x_2$ is large, then the two primitive elements $e_1$ and $e_2$ are not part of the same graph element.

**Rule 4:** If $x_3$ is long, then the two primitive elements $e_1$ and $e_2$ are not part of the same graph element.

Two primitive elements are merged into one graph element if the value from the second fuzzy inference system exceeds a threshold value.

5. **Fitting and Classifying Graph Elements**

Straight lines, circles, ellipses, circular arcs, et cetera, are expressed as basic shapes in the SVG specification; the other curves are expressed by piecewise cubic Bézier curves [3].

Given the finite set of points of an element, denoted by $P = \{ (x_i, y_i) | 0 \leq i \leq n \}$, the method of least squares is applied to $P$ to determine a model: a straight line, a circle, an ellipse, or a circular arc. Here, we minimize the sum of the squared distances between the points and the corresponding points on the model. If the sum is smaller than a threshold value, the set of points is classified as the model.

5.1 **Straight Line, Circle and Arc Classification**

To express a straight line as a SVG code, the two endpoints of the straight line are needed. These endpoints are determined as the two points on the model which are the nearest to the two terminal points of $P$.

The SVG specification requires the center coordinate and the radius to express a circle. First, we calculate the arithmetic mean $\bar{x}$ of the $x$-coordinates, $x_i$s, and also the arithmetic mean $\bar{y}$ of the $y$-coordinates, $y_i$s. Let $u_i = x_i - \bar{x}$ and let $v_i = y_i - \bar{y}$ for $i = 0, 1, \ldots, n$. We solve the problem first in $(u, v)$ coordinates, and then transform back to $(x, y)$ coordinates. Let the circle have the center $(u_c, v_c)$ and radius $r$. We minimize

$$\sum_{i=0}^{n} \left[ (u_i - u_c)^2 + (v_i - v_c)^2 - r^2 \right]^2 .$$

Solving this problem by the method of least squares, we have the center coordinate, $(u_c, v_c)$, and the radius, $r = u_c^2 + v_c^2 + (S_u + S_v)/n$, where $S_u = \sum_{i=0}^{n} u_i^2$ and $S_v = \sum_{i=0}^{n} v_i^2$; the center $(x_c, y_c)$ of the circle in the original coordinates system is $(x_c, y_c) = (u_c, v_c) - (\bar{x}, \bar{y})$.

To find the model of a circular arc, we first find the circle which best fits to $P$, and the endpoints of the circular arc are then determined as the two points on the circle which are the nearest to the terminal points of $P$.

5.2 **Ellipse Classification**

When we write the SVG code of a general ellipse, SVG requires the center coordinate, $(x_c, y_c)$, the gradient, $\alpha$, of the major axis, and the standard equation, $(x/a)^2 + (y/b)^2 = 1$, which is given by applying the two transformations to the general ellipse: the translation which moves the origin $(0,0)$ to the point $(-x_c, -y_c)$ and the rotation whose angle is $\tan^{-1}\alpha$. Let $\bar{x}$ and $\bar{y}$ be the arithmetic means of $x$-coordinates and $y$-coordinates, respectively. We then put the arithmetic mean $(\bar{x}, \bar{y})$ to the center coordinate, $(x_c, y_c)$. Next, Principal Component Analysis [18] is applied to $P$, and the gradient of the first principal component is set to $\alpha$, the gradient of the major axis. After that, the two transformations mentioned above are applied to $P$. Let $(u_i, v_i)$ be the point transformed from $(x_i, y_i)$ by the two transformations. We then solve the least squares problem:

$$\sum_{i=0}^{n} \left[ (b_i^2 u_i^2 + a_i^2 v_i^2 - a_i^2 b_i^2) \right]^2 \rightarrow \text{minimize}$$

5.3 **Cubic Bézier Curve Fitting**

If a graph element is not classified as one of the models above, we then apply an algorithm of piecewise cubic Bézier curve fitting, introduced by Schneider [19], to $P$, the set of points.

A Bézier curve of degree $n$ is defined as

$$Q(t) = \sum_{i=0}^{n} V_i B_i^n(t), \quad t \in [0,1]$$

where the $V_i$’s are the control points, and the $B_i^n(t)$’s are the Bernstein polynomials,
where \( \binom{n}{i} \) is the binomial coefficient. Equation (5) is called a cubic Bézier curve when \( n = 3 \). Fig.6 shows an example of cubic Bézier curves; the four points, \( V_0, V_1, V_2, \) and \( V_3, \) are the control points, the thick curve is the cubic Bézier curve, and the polygon expresses the one constructed by the four control points.

The problem of finding a cubic Bézier curve which best fit the set of points \( P \) is stated in this manner: given a set of points, find a cubic Bézier curve that fits these points within some given tolerance. The fitting criterion here is to minimize the sum of the squared distances from the points to their corresponding points of the curve. Formally, we minimize the function, \( S, \) defined by

\[
S = \sum_{i=0}^{n} (p_i - Q(q_i))^2
\]  

where \( p_i \) is the \( (x_i, y_i) \) coordinates of \( P \) and \( q_i \) is the parameter value associated with \( p_i \). To solve this least squares problem, the following conditions are considered:

1. \( V_0 \) and \( V_3 \), the first and last control points, are given; they are set to be the first and last points of \( P \).

2. Let \( \hat{t}_1 \) and \( \hat{t}_2 \) be the unit tangent vectors at \( V_0 \) and \( V_3 \), respectively. Then, \( V_1 = a_1 \hat{t}_1 + V_0 \) and \( V_2 = a_2 \hat{t}_2 + V_3 \) hold; that is, the two inner control points, \( V_1 \) and \( V_2 \), are each some distance from the nearest end control point, in the tangent vector direction.

Then, our problem can be defined as finding \( a_1 \) and \( a_2 \) to minimize \( S \); that is, we solve the following two equations for \( a_1 \) and \( a_2 \) to determine the inner control points \( V_1 \) and \( V_2 \):

\[
\frac{\partial S}{\partial \alpha_1} = 0 \quad \text{and} \quad \frac{\partial S}{\partial \alpha_2} = 0
\]  

Let \( p \) be the point that is located at the farthest distance from the cubic Bézier curve, which is given by Equation (7). If the distance from \( p \) to the curve exceeds a threshold value, then the set of points \( P \) is divided at point \( p \). Then, equation (7) is applied to both of the two parts until the farthest distance does not exceed the threshold.

6. Experiment Results

Fig.7 shows sample images of mathematical graphs which are selected from mathematics and science textbooks. Then, these graphs were captured by using an image scanner, whose resolution was set at 600 dpi. These electronic images were saved in 24-bit bitmap format. The size of images is about 1,500×1,500 pixels. All the graphs are processed in a binary format; we assume that the images dealt with in this paper are monochromatic (i.e., black for foreground and white for background). We applied these mathematical graphs to our method and examined the effectiveness of the method.

6.1 Results of Broken Line Extraction

Fig.8 shows the experimental results of Fig. 7. In Fig.8, broken line elements of the same cluster are represented in the same color. For the mathematical graphs of Fig.7 (a) − (e) we have correct results, i.e., every broken line of these 4 mathematical graphs has been correctly extracted. On the other hand, we do not have correct results from the mathematical graphs of Fig.7 (f) − (h). We found three major reasons why we have incorrect results, and discuss them below.

1. Let us consider the case for Sample 6. There are two different broken lines in this figure, however our method merged them into a single broken line. This merging is done by the process based on spline functions because there were too many pixels between them.

2. Let us consider the case for Sample 7. This is the opposite case of (1). In Fig.8(g), the broken line, which is part of the ellipse, was classified as the two different broken lines (the blue
and the green). This is because the number of the pixels between the two broken lines is not sufficient for the process based on spline functions to merge them into a single broken line.

(3) Let us consider the case for Sample 8. In this figure, there are two black circles on the two broken lines to emphasize two points \((x, y)\)'s. The red broken line, which includes the upper right circle, was classified as a type 3 chain line. However, the broken line elements, which are next to the red one, were classified as a dotted line. So, our method was not able to find the broken line including the black circle.

We have applied more than 30 mathematical graphs to our method, and examined the effectiveness of the method. The mathematical graph of Sample 6 is the only one graph that two different broken lines are merged into a single broken line. Most incorrect results come from the 2nd case, i.e., two clusters are not merged into a single cluster by the merging process based on spline functions since there are too few pixels between the two clusters.

### 6.2 Results of Solid Line Extraction

For the mathematical graphs of Fig.7 (a), (b), (f) – (h), we have correct results. However, we do not have...
correct results for Fig.7(c) – (e). The reason why we have incorrect results are categorized into two: one is thickness of a curve, and the other is the curvature at an intersection. We discuss reasons why we have the incorrect results below.

(1) There is a small arc inside the circle of Fig.7(e). This arc was divided into two parts at the intersection between the arc and the y-axis (see Fig.8(e)). Since the curvature of the arc at the intersection is large, our method did not merge the two fragments into a single arc.

(2) In the original image, some part of a graph element is emphasized using a thick line (see Fig.7(d)). By the division process from Section 4, we have the two primitive elements: the pink one and the blue one in Fig.8(d). Although they are part of the same graph element, the pink line is much thicker than the blue line, so they were not merged into a single graph element.

(3) In Fig.7(c), the line, \( y_1 = x \), is a tangent to the curve, \( y_3 = \sin x \), at the origin. Fig.9 shows the skeleton of the two graphs around the intersection point of Fig.7(c). In Fig.9, \( e_1 \) and \( e_2 \) are part of the curve, and \( e_1 \) and \( e_3 \) are also part of the line. However, in the skeleton (i.e., Fig.9), the smoothness between \( e_1 \) and \( e_2 \) is degraded, and the smoothness between \( e_1 \) and \( e_3 \) is also degraded. So, the curvature between \( e_1 \) and \( e_2 \) is not small enough to merge them into a single graph element. We see the same degradation in the line between \( e_1 \) and \( e_3 \). Therefore, the tangent and the curve were divided into the three parts, and they did not merge together.

After applying more than 30 mathematical graphs to our method, 6 graphs produced wrong results. Most wrong results comes from the 3rd case.

6.3 Results of Graph Element Fitting

We selected more than 30 mathematical graphs. After performing our method and extracting graph elements, we applied the fitting procedures to the graph elements to find their models. All the models were fitted correctly to the graph elements by visual observation.

6.4 Tactile Graphic Production

Fig.10(a), (b) are tactile graphic images of Sample 1 and 2 in Fig.7. These images have been produced in the following way.

1. The graph elements were extracted by our method.
2. After identifying a natural cubic spline, \( s(x) \), for a graph element, sample points were created by \( s(x) \), and we then determined a cubic Bézier curve.
3. We produced a bitmap image from the cubic Bézier curve so that this bitmap image looks like a tactile graphic produced by a Braille embosser.

As Fig.10 shows, it is fine enough for people with visual impairments to understand the mathematical graphs if the bitmap image was the actual tactile graphic. However, we need to ensure our opinion, that is, we need to check if the visually impaired are able to understand the tactile graphics which are produced by our method. This is one of our future problems.

7. Conclusion and Future Works

We are now developing a system for automating translation of mathematical graphs into tactile graphics. To develop this system, mathematical graph recognition is needed. So, this paper discussed a method for mathematical graph recognition; a method for extracting and classifying graph elements.

We applied more than 30 mathematical graphs to our method, and its effectiveness was examined. Our
method works well for many examples. However, we have incorrect results from some of the examples. Fixing these drawbacks is one of our future endeavors.

The final goal of our research is to convert bitmap images into SVG descriptions. So in the near future, we will create SVG codes to express mathematical graphs.

8. Acknowledgement

This work was supported by JSPS Grant-in-Aid for Scientific Research (C) Number 24501198.

References


[Contact Address]
5180 Kurokawa Imizu, Toyama 939-0398, Japan
Department of Intelligent Systems Design Engineering, Toyama Prefectural University
Jianjun CHEN
TEL: 0766-56-7500 Ext.463
E-mail: t274001@st.tu-oyama.ac.jp
Noboru TAKAGI
TEL: 0766-56-7500 Ext.461
FAX: 0766-56-8030
E-mail: takagi@pu-oyama.ac.jp
Information about Author

Jianjun CHEN  [non-member]

Jianjun CHEN received a B.S. degree from the Shanxi Datong University, Datong, China, in 2007, and a M.S. degree from Dalian Jiaotong University, Dalian, China, in 2010, both in Applied Mathematics. He is currently a Ph.D. student in the Department of Intelligent Systems Design Engineering, Toyama Prefectural University, Toyama Prefecture, Japan. He is currently engaged in research on soft computing and its application to assistive systems for people with disabilities.

Noboru TAKAGI  [member]

Noboru TAKAGI received B.Eng., M.Eng. and Dr.Eng. degrees from Meiji University in 1989, 1991 and 1994, respectively. He is currently an associate professor in the Department of Intelligent Systems Design Engineering, Toyama Prefectural University. His research interests include soft computing and its application to assistive systems for people with disabilities. He is a member of the IEEE Systems, Man, and Cybernetics Society, the Institute of Electronics, Information and Communication Engineers, the Information Processing Society of Japan, and the Japan Society for Fuzzy Theory and Intelligent Informatics.