Information Hiding in SVG by Affine Transformation with Small Perturbations

Chyuan-Huei Thomas Yang
Assistant professor
Department of Computer Science,
Husan-Chuang University,
Hsinchu City, Taiwan 300
chyang@hcu.edu.tw

Tsung-Ming Lin
Graduate Student
Department of Computer Science,
Husan-Chuang University,
Hsinchu City, Taiwan 300
m94301803@umail.hcu.edu.tw

Chin-Chih Chang
Assistant Professor
Department of Computer Science
and Information Engineering,
Chung Hua University, Hsinchu
City, Taiwan 300
changc@chu.edu.tw

Abstract

In this paper we are interested in hiding data in Scalable Vector Graphics (SVG). The SVG is an eXtensible Markup Language (XML) specification and file format to describe static or animated of two-dimensional vector graphics. In steganography area many authors embed secret data into a cover image which may be a binary image, a gray-value image, or a color image. The most effort is to avoid the distortion of the cover image after embedding information. Usually they work on each pixel or a block of the cover image directly. The SVG is one kind of vector graph format that is a plaintext. In SVG the graphs is not a bit map form and is nothing concerning with the resolution. Those methods become useless. We propose a simple and imperceptible data hiding method working on SVG that use the affine transformation with small perturbation. We modify the digits behind the decimal point of the coordinates of the vector graphs in SVG. Then use affine transformation to change the modified digits slightly. This altering is to prevent people observing directly. The modified digits coordinates is insensitive visible with browsing software. The most difficult question is the useful text can be embedded in SVG are very few. To overcome this problem we add a transformation matrix in front of each useful coordinates. We split this matrix into a series consecutive multiple matrices, then embed data during each split. Of course we add a small perturbation after each embedding. We may continue to do this until the whole embedding data is running out. The experiment results demonstrate our method is quite simple and imperceptible.

Keywords: *information hiding, SVG, XML, steganography,, affine transformation*

1. Introduction

Data hiding and watermarking are two popular research topics. Both are concerned with embedding secret data into media such as texts, audios, images, or videos to achieve imperceptibility. They often combine image and signal processing with cryptography, visual perception, and communication. The main purposes of these two kinds of topics are very different. We may say data hiding concentrate on the capacity and the watermarking focus on robustness. Many data hiding techniques have been studied in two main ways that embedded the secret data into either the spatial domain [1~4] or the frequency domain [5~7]. Some authors invent another domain said compression domain [1, 3, 6]. The authors like to use vector quantization (VQ) techniques. They use the difference between the original cover image and the compressed cover image to hide the secret data. In this paper we discuss the data hiding in SVG. Recently not many researchers [8-10] publish their discussion concerning about hiding data or watermarking in SVG.

Firstly, we briefly describe the previous works in spatial domain. Chang and Lin [1] propose a VQ-based data hiding scheme with principle component analysis. They combine vector quantization (VQ) and principle component analysis (PCA). They use the PCA algorithm to sort a VQ codebook where the VQ codebook is generated by the LBG algorithm. Then each codeword in the codebook is closer to its neighbors. Chao et al. [2] apply their proposed method may embed such as diagnostic reports, electrocardiogram, and digital signatures from doctors or a hospital into an electronic patient record (EPR). They use the "bipolar multiplenumber base" technique to combine the functions of authentication, integration, and confidentiality in EPR. The authors propose an

adaptive algorithm to embed data into VQ compressed images. According to the amount of hidden data, this method adaptively varies the embedding process by clustering in the paper of Du and Hsu [3]. They propose a quantization-based method, which uses trellis coded quantization (TCQ). It can be considered as a special case of trellis coding. Due to TCQ employs a set of trellises and set partitioning ideas of trellis coded modulation in order to achieve better distortion performance with low complexity. These are shown in Esen et al. [4]. Secondly, we introduce some authors' works in frequency domain. Ikeda et al. [5], they work on audio data. They use the band-limited random sequences to establish further flexibility in the spread spectrum based audio data hiding. They set up a systematic method in order to generate band-limited and orthonormal random sequences of any length to understand the sub-band data hiding. In Mukherjee et al. [6], the authors present a source and channel coding framework for data hiding. It allows any tradeoff between the distortions, the data embedded, and the robustness. The secret data is source coded by vector quantization. They use orthogonal transform domain vector perturbations to embed data in the cover video after the VQ indices obtaining in the process. LZW is one of the well-known lossless compression methods. The modifiable elements of LZW for data hiding and how to handle them are proposed in Shim et al. [7].

Before reviewing data hiding in SVG papers we introduce SVG first. An XML specification and file format to describe static or animated of two-dimensional vector graphics is called Scalable Vector Graphics (SVG) [8]. SVG can be declared solely or with scripts. SVG may have hyperlinks using simple XLinks. It is an open standard constructed by the World Wide Web Consortium's SVG Working Group. We know SVG observe the XML grammar, and describe the graphs by plaintext. That is one kind of vector graph format and is nothing concerning with the resolution. The SVG format has the following superiors：

1. SVG documents are readable, and easily modified and edited.
2. It can be combined with modern technique, such as the animation of SVG based on Synchronized Multimedia Integration Language (SMIL), and JavaScript can be embedded into SVG document, strictly speaking it should be ECMAScript controls the object in SVG.
3. SVG format is easy to set up the index of the graphs depends on the content inside the graph. It can be achieved by searching the text of the graph's content to find the graph.
4. SVG supports many kinds of filters and special effects. The similar effect of the text's shadows in bit map can be accomplished without altering the graph in SVG.
5. SVG format can be used to generate the graphs dynamically. Such as SVG can dynamically create interactive map and embed into web to display to end users.

The SVG main problem is how to occupy the market of the vector graphs format and compete to the famous and strong opponent Flash. Another problem is what level and how many products support and/or will support the environment to SVG.

The most important references of this paper are concerning with working in SVG [9-11].Zhou, and Pan [9] they use the polygonal line embedding algorithm to parameterize the polygonal lines to a circle of a contour of a region. They use the center of a circle connected to both end points of a piece of polygonal segment line to form an angle that will be used in extraction. These lines are the positions where data are embedded. Yeh and Wu [10] they summarize the possible places to be able to use embedding in SVG. They called their method information mixture. They modified attributes in SVG, such as coordinates, paths, time, color, shape, and matrix. This paper gives us a good idea to build a general hiding method. Wang, Zhuang, and Cheng [11] They convert the secret data into bit stream with shuffling, then collect 4 positions as one group and reorder them to form a watermark. Convert them into octal decimal. By using a k^{th} order polynomial and the time and corresponding position with a modification to form a new SVG.

This paper is organized as follows. In section 2 we discuss our proposed method that includes how to embed the secret data into SVG and extract these embedded data from this marked SVG. We also review the affine transformation briefly. Section 3 then gives several computer experiment results and comments. Finally, in the conclusion we summarize our major results and outline our future work.

2. Proposed method

We propose a simple and imperceptible data hiding method working on SVG that use the affine transformation with small perturbation. We adjust the digits behind the decimal point of the coordinates of the vector graphs in SVG. Applying the affine

transformation slightly modify the embedded digits. This change forbids people to guess the embedding data directly. The modified digits coordinates is insensitive visible during browsing. To triumph over the few useful embedding positions in SVG we add a transformation matrix of each useful coordinates. We split this matrix into a series consecutive multiple matrices, then embed data during each split. Of course we add a small perturbation after each splitting. We keep on doing this until the whole embedding data is running out. Before we describe the proposed method, we briefly review the affine transformation next subsection.

2.1 Affine transformation

An affine transformation is composed of one or more linear transformations (rotation, scaling or shear) and translation (shift). Those affine transformations can be combined into a single matrix. Denote (x, y) is a coordinate, and (x', y') is the corresponding coordinates after applying the affine transformation with (x, y). We describe the affine transformations with the homogeneous coordinates as followings.

Rotation

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \tag{1}$$

where θ is the rotation angle. If R is the rotation matrix, then

$$\boldsymbol{R}^{-1} = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{2}$$

Scaling

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \tag{3}$$

where s_x and s_y are the scaling (enlarging or shrinking) of x coordinate and y coordinate respectively. If S is the scaling matrix, then

$$\boldsymbol{S}^{-1} = \begin{pmatrix} 1/s_x & 0 & 0 \\ 0 & 1/s_y & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{4}$$

Shear

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ m & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \tag{5}$$

where a horizontal shear (or shear parallel to the x axis) for $m \neq 0$ of vertical lines $x = a$ into lines $y = (x - a)/m$ of slope $1/m$ is represented by the linear mapping.

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & m & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \tag{6}$$

where a vertical shear (or shear parallel to the y axis) of lines $y = b$ into lines $y = mx + b$. If M_1 is the horizontal shear matrix and M_2 is the vertical shear matrix, then

$$\boldsymbol{M}_1^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ -m & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{7}$$

$$\boldsymbol{M}_2^{-1} = \begin{pmatrix} 1 & -m & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{8}$$

Translation

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \tag{9}$$

where t_x and t_y are the translation of x coordinate and y coordinate respectively. If T is the translation matrix, then

$$\boldsymbol{T}^{-1} = \begin{pmatrix} 1 & 0 & -t_x \\ 0 & 1 & -t_y \\ 0 & 0 & 1 \end{pmatrix}, \tag{10}$$

We may use some of these linear transformations to create a small perturbation after embedding the date. For simply perform we just use the rotation matrix with its inverse to implement our proposed method.

2.2 Embedding and Extraction

The main idea of the proposed embedding algorithm is to maintain the perceptual of SVG. In our proposed method, we first transform the secret data, the embedded data, into a bit stream. We may use pseudorandom number generator (PRNG) to shuffle this bit stream. A PRNG can be used a seed state then started from an arbitrary starting state. It always produces the same sequence when initialized with that state. We examine the SCV that contains scalable vector graphics format then collect the numbers that represent the coordinates of SVG. These numbers are the potential positions to hide secret data. We may

ignore the digits behind the decimal point of these numbers. We embed a fixed length of decimal digits at the rear of the decimal point of these coordinates after deleting the digits behind the decimal point. The fixed length of decimal digits is the embedded data that obtains from the bit stream. The length of decimal digits determines how many bits to be grabbed each time. We use affine transformation to give those coordinates small perturbations after embedding, since these small perturbations will not cause visual perception during browsing by graphical web browsers and can avoid the embedding data directly observed.

There are two main embedding ways. One is appending directly behind the decimal point of the candidates of coordinates and another is embedding in transform matrix which is in the SVG already or is inserting it manually on purpose. We illustrate here in detail. Let *X* represent a SVG file, and the (x_i, y_i) be the candidates of the embedding places, where i = 1, 2, …, n. We denote *Y* as a secret data, such as an image, a text file or any information can be converted into binary digits to form a bit stream. *X'* is a SVG file with the secret data, called stego-SVG.

Embedding directly to the candidates of coordinates:

If (x_i, y_i) are the candidates of the embedding positions, and δ_1, δ_2 are current embedding data. We delete the decimal digits of x_i and y_i first, then append δ_1 and δ_2 to the x_i and y_i respectively.

For example, in SVG x="400.3" y="100.5" is one of the candidates, we delete the decimal digits first, then it becomes x="400" y="100". We want to embed 13 bits and take from bit stream *Y*, such as δ_1 =0010011010010 and δ_2 =1011000101110. We convert them to δ_1 =1234 and δ_2 =5678. Finally, x="400.1234", y="100.5678" will be the coordinate after embedded. Next we slightly change x and y by an affine transformation, said $\boldsymbol{R} = \begin{pmatrix} \cos 0.01^\circ & -\sin 0.01^\circ & 0 \\ \sin 0.01^\circ & \cos 0.01^\circ & 0 \\ 0 & 0 & 1 \end{pmatrix}$.

This is a rotation matrix with turning around angle 0.01. Thus x= "400.105841513546" y= "100. 637633175353".

To extract the data we use $\boldsymbol{R}^{-1} = \begin{pmatrix} \cos 0.01^\circ & \sin 0.01^\circ & 0 \\ -\sin 0.01^\circ & \cos 0.01^\circ & 0 \\ 0 & 0 & 1 \end{pmatrix}$

Embedding in transform matrix:

If A_i is a transform matrix, then it is $\boldsymbol{A_i} = (\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}, \boldsymbol{e}, \boldsymbol{f})$ in SVG. If $\delta_{i_1}, \delta_{i_2}, \delta_{i_3}, \delta_{i_4}, \delta_{i_5}, \delta_{i_6}$ are embedding data recently, we have $\boldsymbol{A_i} = \boldsymbol{B_i} * \boldsymbol{C_i}$ and $\boldsymbol{B_i} = (2.\delta_{i_1}, 1.\delta_{i_2}, 1.\delta_{i_3}, 1.\delta_{i_4}, 1.\delta_{i_5}, 1.\delta_{i_6})$. Here we separate a matrix into two multiple matrices. There exists a question that one of the multiple matrices after applying affine transformation should be a nonsingular matrix. Otherwise it is unable to find its inverse matrix. We put $2.\delta_{i_1}$ in the first entry of B_i to guarantee the embedded matrix after applied affine transformation is a nonsingular matrix. Thus we may compute $\boldsymbol{C_i}$ by $\boldsymbol{C_i} = \boldsymbol{B_i}^{-1} * \boldsymbol{A_i}$. We apply the affine transformation, i.e. apply a linear transformation or a combination of linear transformations *R* and $\boldsymbol{A_i} = \boldsymbol{B_i} * \boldsymbol{R} * \boldsymbol{R}^{-1} * \boldsymbol{C_i}$, then update $\boldsymbol{B_i} = \boldsymbol{B_i} * \boldsymbol{R}$ and $\boldsymbol{C_i} = \boldsymbol{R}^{-1} * \boldsymbol{C_i}$. The small perturbation is caused by the affine transformation *R*. Here $\delta_{i_1}, \delta_{i_2}, \delta_{i_3}, \delta_{i_4}, \delta_{i_5}, \delta_{i_6}$ are embedding data with proper fixed length decimal digits and they are all less than one.

For example, if A_i=(1,0,0,1,0,0), and $\delta_{i_1} = 123456, \delta_{i_2} = 457812, \delta_{i_3} = 456789,$ $\delta_{i_4} = 134567, \delta_{i_5} = 100123, \delta_{i_6} = 457891$.

Thus

$$\boldsymbol{A_i} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad \boldsymbol{R} = \begin{pmatrix} \cos 0.01^\circ & -\sin 0.01^\circ & 0 \\ \sin 0.01^\circ & \cos 0.01^\circ & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

$$\boldsymbol{B_i} = \begin{pmatrix} 2.123456 & 1.456789 & 1.100123 \\ 1.457812 & 1.134567 & 1.457891 \\ 0 & 0 & 1 \end{pmatrix}.$$

We apply *R* to B_i, B_i becomes

$$\boldsymbol{B_i} = \begin{pmatrix} 2.12371022530219 & 1.45641836482650 & 1.10012300000000 \\ 1.45800999709259 & 1.13431254652810 & 1.45789100000000 \\ 0 & 0 & 1 \end{pmatrix}.$$

$$\boldsymbol{C_i} = \boldsymbol{B_i}^{-1} * \boldsymbol{A_i}$$

$$C_i = \begin{pmatrix} 3.97337129812296 & -5.10167232706127 & 3.06648501796677 \\ -5.10724763871819 & 7.43912185453108 & -5.22682820557460 \\ 0 & 0 & 1 \end{pmatrix}.$$

The original SVG has transform= "(1,0,0,1,0,0)". It becomes transform = "matrix(2.12371022530219, 1.45800999709259, 1.45641836482650, 1.13431254652810, 1.10012300000000, 1.45789100000000) matrix(3.97337129812296, -5.10724763871819,

-5.10167232706127, 7.43912185453108, -3.06648501796677, -5.22682820557460)"

The computer experiment of this example will be shown in next section. We present both embedding algorithm and extraction algorithm as following. Also the flowcharts of embedding algorithm and extraction algorithm are shown in Fig. 1 and Fig.2

Embedding Algorithm

Step 1: Convert Y into a bit stream.

Step 2: Shuffle Y by PRNG.

Step 3: Examine X to find the (x_i, y_i) are the candidates of the embedding positions and collect them.

Step 4: If the size of Y is too large to embed into those candidates, Go to Step 8.

Step 5: According to the size of bit stream Y to decide how many bits of each time to embed. Convert the proper length of bits into decimal digits. Append these convert decimal digits to the (x_i, y_i) of each coordinate.

Step 6: Applying the affine transform matrix R, such as using rotation matrix by rotating θ to give a small perturbation,

Step 7: If embedding data are not running out, go to Step 5,
else embed into transform matrix A, go to Step 7,
if embedding data are not running out, go to Step 8.
else Stop.

Step 8: We put a transform matrix I_i = (1,0,0,1,0,0), i=1, before one of candidates (x_i, y_i) in SVG.

Step 9: We may estimate how many times that we have to split transform matrix into two multiple matrix.

Step 10: Let
$\boldsymbol{I}_i = \boldsymbol{B}_i * \boldsymbol{C}_i$,
$B_i = \left(2.\delta_{i_1}, 1.\delta_{i_2}, 1.\delta_{i_3}, 1.\delta_{i_4}, 1.\delta_{i_5}, 1.\delta_{i_6}\right)$,
and $\boldsymbol{I}_i = \boldsymbol{B}_i * \boldsymbol{R} * \boldsymbol{R}^{-1} * \boldsymbol{C}_i$, the new $\boldsymbol{B}_i = \boldsymbol{B}_i * \boldsymbol{R}$ and the new $\boldsymbol{C}_i = \boldsymbol{R}^{-1} * \boldsymbol{C}_i$,
where $\delta_{i_1}, \delta_{i_2}, \delta_{i_3}, \delta_{i_4}, \delta_{i_5}, \delta_{i_6}$ are embedded data with proper fixed length decimal digits and they are all less than one.

Step 11: If embedding data are running out to get the stego-SVG X' ,
then Stop,
else if no space to embed
i=i+1, go to Step 8,

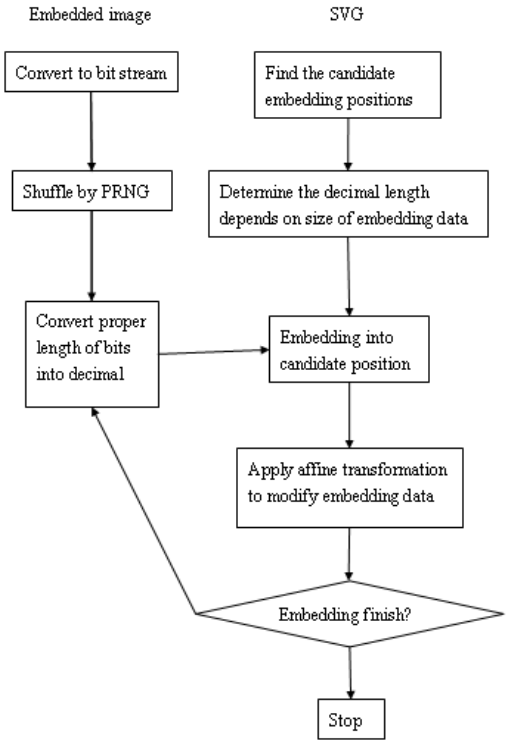

Fig. 1: the flowchart of embedding algorithm

Extraction Algorithm

Step 1: Examine X' to find the (x_i, y_i) are the candidates of the embedding positions and collect them.

Step 2: If there is no transform matrix, collect the digits behind decimal point of x_i, and y_i then convert it to binary digits and append them to a bit stream, until extracting finish. Go to Step 4.

Step 3: If we have transform = "matrix $\boldsymbol{B}_1$ matrix $\boldsymbol{C}_1$ matrix $\boldsymbol{B}_2$ matrix $\boldsymbol{C}_2$ ….. matrix $\boldsymbol{B}_k$ matrix $\boldsymbol{C}_k$", and let $\boldsymbol{B} = \boldsymbol{B}_1$, then apply the inverse affine transform matrix R^{-1}, i.e. use $\boldsymbol{B}_i * \boldsymbol{R}^{-1}$ to extract the six positions' pure decimal digits. Collect all six decimal digits, convert them to binary digits. Append them to a bit stream, until extract all such transform matrix finishing.

Step 4: Shuffle bit stream back by PRNG.

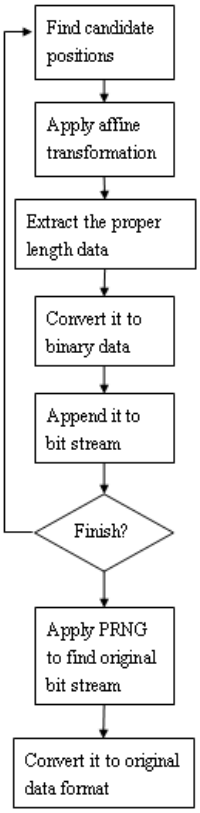

Fig. 2: the flowchart of extraction algorithm

3. Experiment Results

Here we give two examples to show embedding directly to the candidates of coordinates.

Example 1: rectangle.svg

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
    xmlns="http://www.w3.org/2000/svg" version="1.1">
<rect x="400.3" y="100.5" width="400" height="200"
      fill="yellow" stroke="navy" stroke-width="1" />
</svg>
```

The vector graphic is shown by SVG browser as below in Fig. 3.

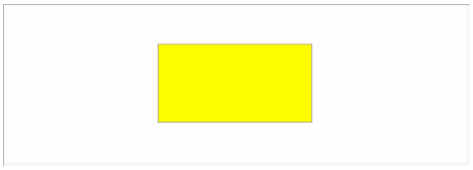

Fig. 3: the vector graph of rectangle.svg

We embed (5, 7) that are converted to binary (101 , 111) . (x="400.3" , y="100.5") become (x="400.5" , y="100.7"). We apply the affine transformations that rotation angle is 0.01°, the translation of *x* coordinate is 0.01, and the translation of *y* coordinate also is 0.01. New coordinates of (x, y) = (400.482418434558, 100.769898902439). The SVG file is modified as below.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"

"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
    xmlns="http://www.w3.org/2000/svg" version="1.1">
<rect x="400.482418434558" y="100.769898902439" width="400" height="200"
      fill="yellow" stroke="navy" stroke-width="1" />
</svg>
```

We have Fig. 4 that is almost the same as Fig. 3 shown as following.

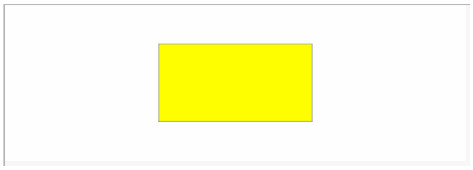

Fig. 4: the vector graph of "rectangle.svg" after embedded.

We give another example to show embedding directly after the candidates.

Example 2: rectangle_circle.svg

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"

"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
```

```
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
   xmlns="http://www.w3.org/2000/svg" version="1.1">
 <rect x="400.3" y="100.5" width="400" height="200"
      fill="yellow" stroke="navy" stroke-width="1" />
 <circle cx="600.5" cy="200.5" r="100"
      fill="red" stroke="blue" stroke-width="1" />
</svg>
```

The vector graphic is shown by SVG browser as below in Fig. 5.

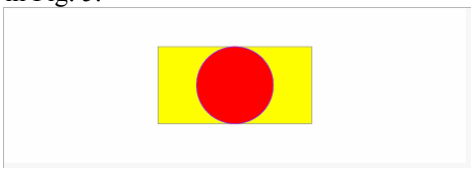

Fig. 5: the vector graph of rectangle_ circle.svg

We embed (135, 579)，(246，468) that are converted to binary (10000111，1001000011)；(11110110，111010100) . (x="400.3", y="100.5") and (cx="600.5" cy="200.5") become (x="400.135", y=", 100.579") and (cx="600.246" cy="200.468"). We apply the affine transformations that rotation angle is 0.01°, the translation of *x* coordinate is 0.01, and the translation of *y* coordinate also is 0.01. New coordinates of (x, y) = (400.117439558601, 100.648835199764) and (cx, cy) = (600.211002591459, 200.572759636385). The SVG file is modified as below.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"

"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
   xmlns="http://www.w3.org/2000/svg" version="1.1">
  <rect x="400.117439558601" y="100.648835199764" width="400" height="200"
      fill="yellow" stroke="navy" stroke-width="1" />
  <circle cx="600.211002591459" cy="200.572759636385" r="100"
      fill="red" stroke="blue" stroke-width="1" />
</svg>
```

The graph is the similar to the Fig.3 shown in Fig. 6.

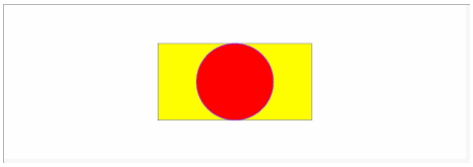

Fig.6: the vector graph of rectangle_ circle.svg after embedded.

We use an example to embed a series of fixed length digits in transform matrix. In this example there is no transform matrix. We need to set an identity matrix as a transform matrix, thus we may split it to embed data.

Example 3: two_rectangle.svg

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"

"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
   xmlns="http://www.w3.org/2000/svg" version="1.1">
 <!-- Show outline of canvas using 'rect' element -->
 <rect x="1" y="1" width="1198" height="398"
      fill="none" stroke="blue" stroke-width="2"/>
 <rect x="400" y="100" width="400" height="200"
      fill="yellow" stroke="navy" stroke-width="10" />
</svg>
```

The vector graph is shown in Fig. 7.

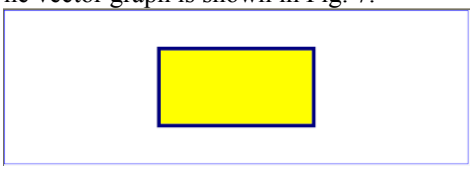

Fig. 7: the vector graph of two_rectangle.svg

We have described this example in embedding in transform matrix last section. Actually this example is no such transform matrix. We put a virtual transform matrix to create six embedding positions. Therefore, the SVG may be modified as follows.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="12cm" height="4cm" viewBox="0 0
```

```
1200 400"
    xmlns="http://www.w3.org/2000/svg"
version="1.1">
 <!-- Show outline of canvas using 'rect' element -->
  <rect transform="matrix(2.12371022530219,
1.45800999709259, 1.45641836482650,
1.13431254652810, 1.10012300000000,
1.45789100000000)
matrix(3.97337129812296, -5.10724763871819,
-5.10167232706127, 7.43912185453108,
-3.06648501796677, -5.22682820557460)
"  x="1" y="1" width="1198" height="398"
       fill="none" stroke="blue" stroke-width="2"/>
  <rect x="400" y="100" width="400" height="200"
       fill="yellow" stroke="navy" stroke-width="10"
/>
</svg>
```

Of course, we have Fig. 8 that small changes are invisible.

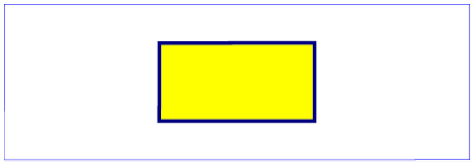

Fig.8: the vector graph of two_rectangle.svg after embedded.

We have two kinds of embedding data into transform matrix in SVG. If the SVG has transform matrix already, we just use it to split two matrices to embed one to six positions. If there is no such transform matrix, we put an identity matrix and embed the data like last example. If there still has data left, we may put the identity matrix as many as we want after the previous identity matrix. Of course we have to follow the embedding steps.

4. Conclusions

We propose a simple and imperceptible data hiding method working on SVG using the affine transformation with small perturbation. We modify the digits behind the decimal point of the coordinates of the vector graphs in SVG. The SVG is shortage of possible embedding positions, since it is plaintext format. We use splitting matrices from transform matrix to embed secret data to overcome this deficit. The results show they are good enough to prevent the perception.

5. References

[1] C.C. Chang, P.Y. Lin "A compression-based data hiding scheme using vector quantization and principle component analysis", Proceedings of the 2004 International Conference on Cyberworlds (CW'04).

[2] Hui-Mei Chao, Chin-Ming Hsu and Shaou-Gang Miaou "A data-hiding technique with authentication, integration, and confidentiality for electronic patient records," IEEE TRANSACTIONS ON INFORMATION TECHNOLOGY IN BIOMEDICINE, VOL. 6, NO. 1, MARCH 2002, pp46-53.

[3] W.-C. Du and W.-J. Hsu "Adaptive data hiding based on VQ compressed Images" IEE Proceedings Visual Image Signal Processing, Vol. 150, No. 4, August 2003, pp233-238.

[4] E. Esen, **A. A.** Alatan, and M. ASkar, "Trellis coded quantization for data hiding" EUROCON **2003** Ljubljana, Slovenia, pp384-388.

[5] Mikio IKeda, Kazuya Tkeda and Fumitada Itakura "Audio Data Hiding by Use of Band-limited Random Sequence" 1999 IEEE Conference, pp 2315-2318.

[6] D. Mukherjee, J. J. Chae, S. K. Mitra, and B. S. Manjunath "A Source and Channel-Coding Framework for Vector-Based Data Hiding in Video" IEEE Transaction on Circuits and Systems for Video Technology Vol. 10, No. 4 June 2000, pp 630-644.

[7] Hiuk Jue Shim, Jinhueng Ahn, and Byeitngwoo Jeon "DH-LZW: Lossless Data Hiding in LZW Compression" 2004 International Conference on Image Processing (ICIP 2004), Singapore, October 24-27, 2004, pp 2195-2198.

[8] Wikipedia, the free encyclopedia http://en.wikipedia.org/wiki/Scalable_Vector_Graphics

[9] Xu Zhou, X. Pan "Watermark-Based Scheme to Protect Copyright of SVG Data" Computational Intelligence and Security, 2006 International Conference on Volume 2, Issue , 3-6 Nov. 2006 pp 1199-1202

[10] Y. Yeh, P. W, "Research of SVG Images Information Hiding" 2005 NCS, Kun Shan University, Tainan.

[11] Y. Wang, C. Zhuang, X. Cheng, "Applcation of digital watermark in SVG information Safety" Journal of South west University for Nationalities, Jan. 2005, pp 156 -159