RAWVec – A Method for Watermarking Vector Maps *

Douglas Aurélio Marques[1], Karina M. Magalhães[2], Ricardo R. Dahab[2]

[1] CGU - Controladoria-Geral da União
SAS Q1 BL. A - 70070-905 - Brasília - DF - Brazil

[2]Instituto de Computação – Universidade Estadual de Campinas (UNICAMP)
Caixa Postal 6176 – 13084-971 – Campinas – SP – Brazil

douglas.marques@cgu.gov.br, karina.magalhaes@students.ic.unicamp.br,

rdahab@ic.unicamp.br

Abstract. *The information used in geographic information system (GIS) and in spatial data is represented by digital vector maps, which are expensive to produce, but easy to copy. Watermarks have been used for a long time in other digital media for both authentication and tracing. This work presents a new method for embedding watermarks, in the form of a bitmap image, into digital vector maps. The detection of the watermark is accomplished by extracting the embedded image and comparing it with the original one.*

1. Introduction

Vector maps represent images using geometric structures like points and circles. They are used in geographic databases for a large variety of ends and services. It is important to counter illegal copying and distribution of these digital contents because they are expensive to produce, but easy to copy. Watermarks have been used for a long time in other digital media (like audio, video and bitmap images) for authentication or tracing. In recent years, however, watermarking vector maps has received more attention, especially due to the popularity of geographic information systems (GIS) and spatial data on the Web.

Usually the methods for watermarking vector maps use the map's objects' coordinates as the basic parameters to embed the watermark, which may result in changes on the vector map. As seen in [Sion 2002], attacks that change the watermark significantly also may change the vector map. This is a big issue in commercial vector maps that usually have a tolerable maximum error, a measure of its trustworthiness.

Most methods, like [Ohbuchi et al. 2003b], [Ohbuchi et al. 2003a] and [Voigt et al. 2004], only cover similarity transformations attacks, ignoring more complex attacks like data changes and projection systems. Moreover, changes on the objects' topology when the watermark is embedded are also ignored. The embedding algorithm must maintain objects' properties like area, size, shape and connectivity. In [Shao et al. 2005] and [Ohbuchi and Masuda 2000], the objects' shape is preserved and in [Praun et al. 1999], the objects' connectivity is preserved. Other methods for watermarking vector maps can be found in [Gou and Wu 2004], [Voigt and Busch 2002], [Voigt et al. 2004] and [Sonnet et al. 2003].

*Second author supported by FAPESP scholarship number 2006/05219-7

In this paper, we provide a new method for watermarking vector maps (Raster Watermarks in Vector Maps, or simply RAWVec), as proposed in [Marques 2005]. The main feature of our method is that the watermark is represented by a raster image, rather than a binary sequence. Therefore, the watermark verification can be performed by human eye (besides a traditional probabilistic algorithm), which increases the effectiveness of our method.

This paper is organized as follows: in Section 2 we describe the embedding and detection algorithms; in Section 3 we analyze the method and discuss its robustness against several attacks. Finally, in Section 4, we present the conclusions and discuss future work.

2. Algorithm Description

In this Section we describe the RAWVec (Raster Watermarks in Vector Maps) method. The watermark used in this method is a raster image: a bitmap image represented by a bi-dimensional matrix whose elements store the color information or intensity for each pixel.

The embedding algorithm marks the vector map with the raster image by shifting the geometric structures' coordinates. The detection algorithm uses the original vector map to extract a raster image that should be compared with the original one. Both algorithms use, besides the raster image R and the original vector map M, a constant C and two functions $w()$ and $v()$ described below.

The function $w : A_{n\times n} \rightarrow A_{n\times n}$ modifies a matrix A by shifting its elements as follows: Let A be a square matrix of order n; then $B = w(A)$, with $b_{ij} = a_{uv}$, where $u = n - i + 1$ and $v = n - j + 1$.

Note that $w(w(A)) = A$. Figure 1 shows an example of the application of function w.

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix} => w(A) = \begin{pmatrix} 16 & 15 & 14 & 13 \\ 12 & 11 & 10 & 9 \\ 8 & 7 & 6 & 5 \\ 4 & 3 & 2 & 1 \end{pmatrix}$$

Figure 1. An example of function $w(A)$

Function v calculates the point representation P from a vector map M. Vector maps are formed by geometric structures and based on these structures' points it is possible to create the point representation $P = v(M)$. Thus, each structure is decomposed into points which are stored according to their type:

- Punctual objects: Structures described by points, like symbols and text. The coordinates of each point are stored.
- Linear objects: Structures described by points' sequences, like lines and polygons. The coordinates of each point in the sequence are stored.
- Parameterized objects: Structures described by parameters, like circles and ellipses. The object can be segmented in a few points, and these are stored; or the main points can be stored. If the object is a circle, for example, we can store a few points of the circle, or only the center.

A point must not be stored more than once, even if it belongs to more than one object. Furthermore, each coordinate must have a reference to its original objects, making it possible to rebuild all the structures. An example of the application of function v is shown in Figure 2.

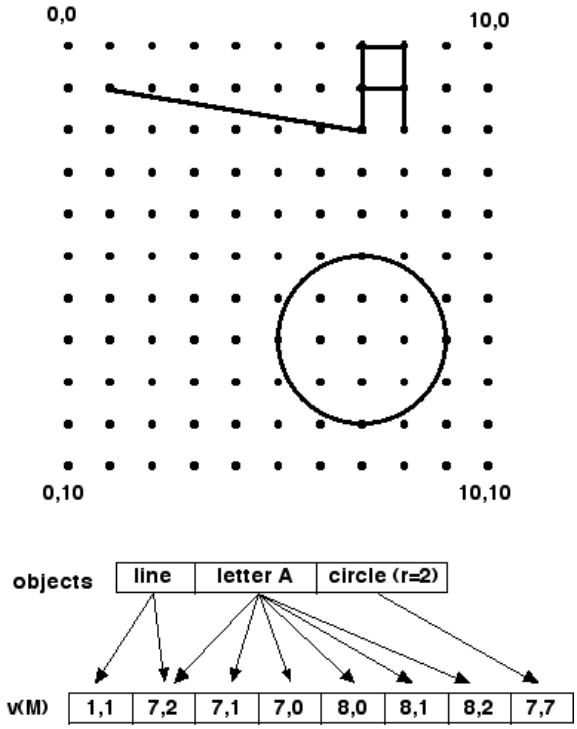


Figure 2. An example of the application of the function v

2.1. Watermark Embedding

Algorithm 1 below embeds a raster image R in the vector map M producing the marked vector map M'. It used a constant C, a positive real constant that controls the maximum shifting of a pixel, so it will stand below the maximum tolerable error of the vector map. We discuss more about this on Section 3.1. The four basic steps of this algorithm, as seen in Figure 3, are:

Algorithm 1 Watermark Embedding

INPUT: raster image R, vector map M and constant C.
OUTPUT: marked vector map M'.

1. Calculate the point representation $P \leftarrow v(M)$ of vector map M.
2. Obtain matrices A_x and A_y from point representation $v(M)$.
3. Resize raster image R, creating the new raster image E.
4. Calculate $B_x \leftarrow CE + A_x$; $B_y \leftarrow Cw(E) + A_y$, for C a real constant.
5. Build the marked vector map using matrices B_x and B_y.

Steps 1 and 2 build two square matrices – A_x for the x coordinate and A_y for the y coordinates – of point representation $P = v(M)$. Let t be the number of points stored in P, n be the order of the matrices and $p_i = (x_i, y_i)$ be the points stored in P. Then

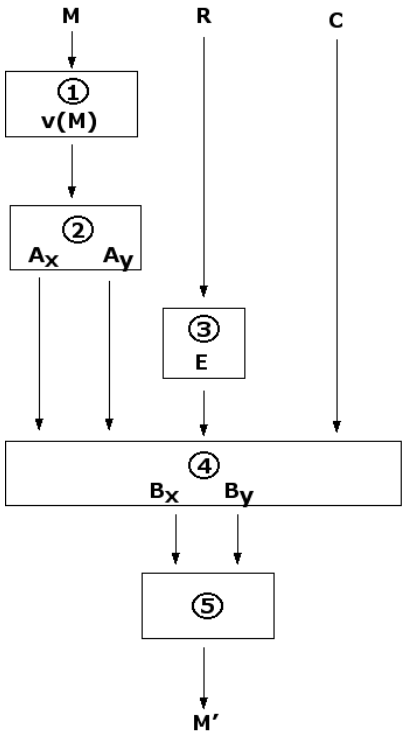


Figure 3. Watermark Embedding Algorithm Diagram

$$n = \sqrt{t}, \tag{1}$$

$$(A_x)_{ij} = x_{n(i-1)+j} \quad \text{and} \quad (A_y)_{ij} = y_{n(i-1)+j} \tag{2}$$

After obtaining matrices A_x and A_y from vector map M, we must resize the raster image R, creating a new raster image E that has $n \times n$ pixels, as illustrated in Figure 4.

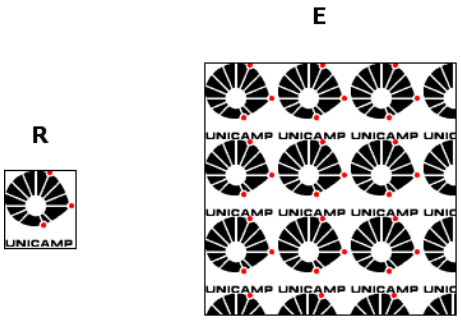


Figure 4. Resizing a raster image R, creating a new raster image E

Now that matrices A_x and A_y and the new raster image E have been calculated, we are able to embed the watermark, calculating B_x and B_y using the equations (3) and (4), respectively:

$$B_x = CE + A_x, \tag{3}$$

$$B_y = Cw(E) + A_y. \tag{4}$$

Finally, the marked vector map M' is built using matrices B_x and B_y.

2.2. Watermark Detection

The detection algorithm consists in the extraction of the watermark and its comparison with the original one. It uses the original vector map M, the constant C and the original watermark R to extract the watermark S from the target vector map N, the one that is going to be tested. The basic steps are represented in Figure 5 and are described below as Algorithm 2.

Algorithm 2 Watermark detection

INPUT: raster image R, vector map M, constant C and the vector map to be tested N.
OUTPUT: watermark S.

1. Embed the watermark R on the vector map M by using the algorithm described in Section 2.1, returning the marked vector map M'.
2. Calculate the point representation $P \leftarrow v(M')$ of map M' and the point representation $Q \leftarrow v(N)$ of target map N.
3. Compare the point representations P and Q using *Point Pattern Matching* (Section 2.2.1), returning the point lists L_p, L_q and a transformation T, where $M' = T(N)$.
4. Build matrices A_x and A_y from M, and B_x and B_y from $T(N)$.
5. Calculate $D_x \leftarrow \frac{B_x - A_x}{C}$; $D_y \leftarrow \frac{w(B_y) - w(A_y)}{C}$.
6. Calculate $D \leftarrow \frac{D_x + D_y}{2}$.
7. Resize the watermark D to its original size, returning the watermark S.

Most attacks consist in transformations on the marked vector map M', producing a vector map N which may not produce the original watermark. Therefore, it is important to find the transformation T used in the attack and remove it from the vector map N. The *Point Pattern Matching* Algorithm finds relations between the points in two vector maps, looking for transformations T that may have been made in one of them.

The first step of the detection algorithm is to embed the watermark R on the vector map M, resulting on the marked map M'. This marked vector map must be compared with the vector map N to be tested, using the *Point Pattern Matching* Algorithm. The point representations from both maps are calculated: $v(M') = P$ and $v(N) = Q$, as described in Section 2 and the *Point Pattern Matching* Algorithm is used to find relations between them. This step is described in Section 2.2.1 and returns two points representations L_p and L_q from vector maps M' and N, respectively, and the transformation T which takes M' to N. Thus $M' = T(N)$.

Now we are able to build four matrices A_x and A_y from $v(M)$ and B_x and B_y from $v(T(N))$. Matrices A_x and A_y represent the original vector map and matrices B_x and B_y represent the vector map $T(N)$ that may be M'. So, we must calculate the watermark in vector map $T(N)$ and compare it with the original one. The watermark D is calculated using the four matrices A_x, A_y, B_x and B_y, the constant C and the function w:

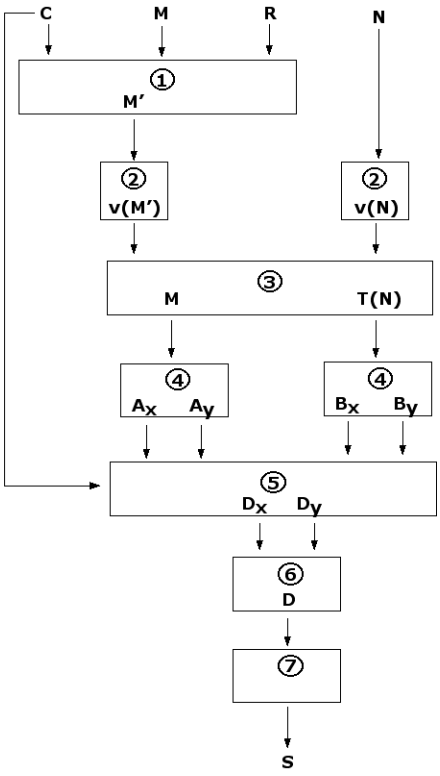


Figure 5. Watermark detection Algorithm Representation

$$D = \frac{D_x + D_y}{2}, \tag{5}$$

where

$$D_x = \frac{B_x - A_x}{C} \quad \text{and} \quad D_y = \frac{w(B_y) - w(A_y)}{C}. \tag{6}$$

Finally, the watermark D, with size $n \times n$, must be resized to its original size – the size of watermark R – producing watermark S. Watermarks S and R must be compared using a probabilistic algorithm and the human eye. According to this comparison we can conclude whether or not the vector map N was marked.

2.2.1. Point Pattern Matching

The ith point in $v(M)$ matches the ith point in $Q = v(M')$. Therefore, if the vector map N is in fact the vector map M', then the ith point in $v(M)$ must match the ith point in $P = v(N)$. However, the vector map N may be obtained from a transformation T in M' and the relation between the points may not be so simple. The *Point Pattern Matching* Algorithm finds this relation between points and returns the transformation T that may have been used in M' to produce N. In this paper we use the *Point Pattern Matching* Algorithm [van Wamelen et al. 1999] described in Algorithm 3.

The *Point Pattern Matching* Algorithm finds a transformation that produces (q, b) from (p, a), tests it in a set of points and finally checks it with all the points. Therefore,

Algorithm 3 Point Pattern Matching

INPUT: point representations $P = v(N)$ and $Q = v(M')$.
OUTPUT: transformation T and two points representations L_p and L_q.

1. Obtains the k nearest neighbors (KNN) for each point in P and Q using a *Delaunay Tree*, as described in [Boissonnat and Teillaud 1993]
2. $global \leftarrow false$;
3. **while** $global = false$
 3.1. **for** each point $p \in P$
  3.1.1 Obtain the nearest point $q \in Q$
  3.1.2 Find (a_f, b_f), the furthest of the KNN, with $a_f \in$ KNN(p) and $b_f \in$ KNN(q)
  3.1.3 Calculate the local transformation T that, used on (p, a_f) produces (q, b_f)
  3.1.4 **if** $T(p, a) = (q, b)$ for at least $\rho\, k$ pairs (a, b) **then**
   3.1.4.1 **if** T is a global transformation **then**
    3.1.4.1.1 $global \leftarrow true$;
    3.1.4.1.2 return (T);

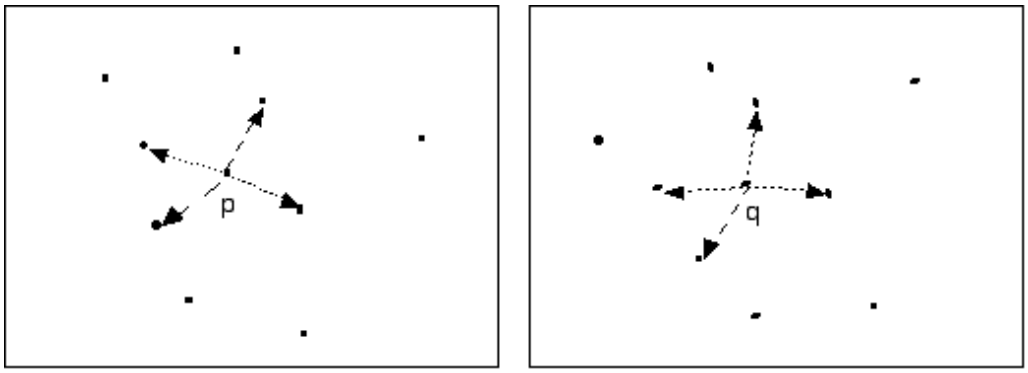


Figure 6. *Point Pattern Matching* Example

the nearest neighbors (KNN) for each point in P and Q must be obtained, using a *Delaunay Tree*, as described in [Boissonnat and Teillaud 1993]. Moreover, we need to find the nearest point $p \in P$ for each point $q \in Q$ and the furthest neighbors a and b, respectively, for both points, p and q.

The next step is to find the local transformation T that when applied to a produces b and when applied to p produces q, that is $T(a) = b$ and $T(p) = q$, with:

$$T = \begin{pmatrix} t_x \\ t_y \end{pmatrix} + s \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}, \tag{7}$$

where

$$s = \frac{|\vec{qb}|}{|\vec{pa}|}, \tag{8}$$

$$\theta = \text{ angle between } \vec{pa} \text{ and } \vec{qb}, \tag{9}$$

and

$$t_x = q_x - p_x s\cos\theta + p_y s\sin\theta \quad \text{and} \quad t_y = q_y - p_x s\sin\theta - p_y s\cos\theta \tag{10}$$

Then this transformation must be checked to see if it produces a subset of KNN(q), if used in a subset of KNN(p) with at least ρk points. These two subsets are two points representations called L_a and L_b.

Now, we must check whether the local transformation T is a global transformation, by calculating the transformation T' and applying it on every point $q \in Q$. If $T'(q)$ produces a point $p \in P$, then p and q must be included in two point lists L_p and L_q, respectively.

$$T' = \begin{pmatrix} t_x \\ t_y \\ s\cos\theta \\ s\sin\theta \end{pmatrix} = \frac{1}{\mu_d} \begin{pmatrix} \mu_a & 0 & -\mu_{ax} & \mu_{ay} \\ 0 & \mu_a & -\mu_{ay} & \mu_{ay} \\ -\mu_{ax} & -\mu_{ay} & l & 0 \\ \mu_{ay} & -\mu_{ax} & 0 & l \end{pmatrix} \begin{pmatrix} \mu_{bx} \\ \mu_{by} \\ \mu_{a+b} \\ \mu_{a-b} \end{pmatrix}, \tag{11}$$

where

$$\mu_{ax} = \sum_{i=1}^{l} L_{ax_i} \quad \text{and} \quad \mu_{ay} = \sum_{i=1}^{l} L_{ay_i}, \tag{12}$$

$$\mu_{bx} = \sum_{i=1}^{l} L_{bx_i} \quad \text{and} \quad \mu_{by} = \sum_{i=1}^{l} L_{by_i}, \tag{13}$$

$$\mu_{a+b} = \sum_{i=1}^{l} L_{ax_i} L_{bx_i} + L_{ay_i} L_{by_i}, \tag{14}$$

$$\mu_{a-b} = \sum_{i=1}^{l} L_{ax_i} L_{by_i} - L_{ay_i} L_{bx_i}, \tag{15}$$

$$\mu_a = \sum_{i=1}^{l} L_{ax_i}^2 L_{ay_i}^2, \tag{16}$$

and

$$\mu_d = l\mu_a - \mu_{ax}^2 - \mu_{ay}^2. \tag{17}$$

If L_p and L_q have at least ρt points (with t being the size of P and Q) the transformation T is global; if not, another pair (p, q) and another transformation T must be found. If all points in P and Q were used and no global transformation was found, then N was not produced from M' and, therefore, it is not marked.

3. Experiments and Results

3.1. Algorithm Analysis

Let t be the number of points on the original vector map; then, the RAWVec method has complexity $O(t)$ for the embedding algorithm and $O(t(\log\, t)^{3/2})$ for the detection algorithm. This is because each operation of the embedding and detection algorithms, except for *Point Pattern Matching*, has complexity $O(t)$. On the other hand, *Point Pattern Matching* has complexity $O(t(\log\, t)^{3/2})$, as proved in [van Wamelen et al. 1999].

Let C be a constant and I_{max} and I_{min} be the maximum and minimum intensities of pixels in the raster image. Then the maximum shifting of a pixel is $\pm CI\sqrt{2}$, with $I = I_{max} - \frac{I_{min}+I_{max}}{2}$, as seen in Figure 7. Therefore, it is possible to control the shifting of each point using the constant C and the intensity of each pixel on the raster image, but it is important to normalize the raster image, changing the intensity of each pixel r:

$$I_{r_{new}} = I_{r_{old}} - \frac{I_{min} + I_{max}}{2}. \tag{18}$$

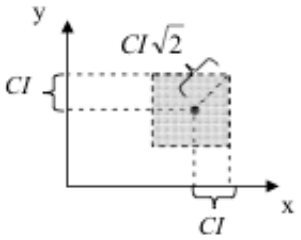


Figure 7. Maximum shifting of a pixel

The comparison between the original watermark R and the extracted watermark S uses two coefficients: r, the Pearson correlation coefficient and h, the quality coefficient. The quality coefficient h is based on human observation. It can vary from 0 to 5 and it represents the answer from 5 people to the question "Do you think that the image R is the image D after some modifications?" Each positive answer represents 1 and each negative answer represents 0, and h is the sum of the answers. The Pearson correlation coefficient r is based on the pixels intensities and it is calculated as follows:

Let I_{m_R} and I_{m_S} be the average intensity of images R and S, respectively, and $I_{r_{ij}}$ and $I_{s_{ij}}$ be the intensity of pixels $r_{ij} \in R$ and $s_{ij} \in S$, respectively. Then

$$r = \frac{\sum_i \sum_j (I_{r_{ij}} - I_{m_R})(I_{s_{ij}} - I_{m_S})}{\sqrt{(\sum_i \sum_j (I_{r_{ij}} - I_{m_R})^2)(\sum_i \sum_j (I_{s_{ij}} - I_{m_S})^2)}}. \tag{19}$$

3.2. Attacks

Attacks against a watermark consist in changing the marked vector map, making it difficult for the watermark to be recognized. The most common attacks that can be used against a watermark are:

- Transformation: Translation, rotation and downscaling are similarity transformations that can be used on the marked vector map and that affect the watermark. They are simple transformations and can be removed using *Point Pattern Matching*. However, downscaling transformations can affect the watermark even if removed, due to lack of precision. Datum and projection changes are more complex transformations that can also be used to affect the watermark. *Point Pattern Matching* does not detect them and the RAWVec Algorithm is not resilient to this kind of attack.

- Cropping: The marked vector map can be cropped or some objects can be deleted. *Point Pattern Matching* can handle cropping because it ignores points and objects from the original vector map that do not have a match in the marked vector map. However, if the cropping is too intense, the watermark can be more affected because each lost point implies in a "hole" in the extracted watermark. If the raster image is smaller than the vector map, copies of it will be used as watermark (Figure 4), and information lost in one copy can be found in another one, making it possible to rebuild and recognize the image.
- Object insertion: *Point Pattern Matching* can also handle object insertion, because it ignores points in the marked vector map that do not have a match in the original vector map. Therefore, the watermark will not be affected by this kind of attack.
- Object order scrambling: *Point Pattern Matching* Algorithm is capable of matching the right objects from the two vector maps (the original and the marked one) even if they differ on the order they appear in the maps.
- Addition of random noise: This attack can be done by adding random numbers to each coordinate of the vector map, and its amplitude is the maximum number added. If the amplitude is above the maximum tolerable error, the vector map will be modified and become useless.
- File format change: There are several commercial file formats and it may be useful to convert one into another. In some cases, the conversion can modify too much the vector map, making it useless. Usually, objects are deleted, inserted or its order is scrambled in the conversion. *Point Pattern Matching* can handle this problem, because it finds the matching between the objects in the two vector maps, ignoring points with no match.

Several tests using these attacks were made. Here we present only some of them (more can be found on [Marques 2005]). Figure 8 shows a simple test, in which there are no attacks; Figure 9 shows a test with a combined attack, in which the vector map is downscaled to one-tenth of its original size, rotated 232^o, translated and finally cropped to half; and Figures 10 and 11 show tests with a cropping attack. Specifically, Figure 10 shows the extracted watermarks for different cropping attacks, and Figure 11 shows the attacked vector map for the particular case in which 90% of the points were removed. Note that, for the cropping attack, the watermark detection failed only when the vector map was clearly changed and became useless.

4. Conclusions and Future Work

In this paper, a new method for watermarking vector map (RAWVec) is presented. This method embeds a raster image in the vector map as the watermark and shifts the map's objects' coordinates. The detection algorithm extracts a watermark from a vector map using the original vector map. The extracted watermark is compared with the original one both algorithmically and using the human eye, increasing its effectiveness. Attacks can alter or delete pixels from the watermark, but the human eye can easily recognize the image. The RAWVec method is robust against several attacks including cropping, addition of random noise, object order scrambling and similarity transformation.

Future works include improving the RAWVec method by making it public, that is, the original vector map would not be necessary on the detection algorithm. Therefore, the use of the *Point Pattern Matching* Algorithm would be avoided, which may improve the

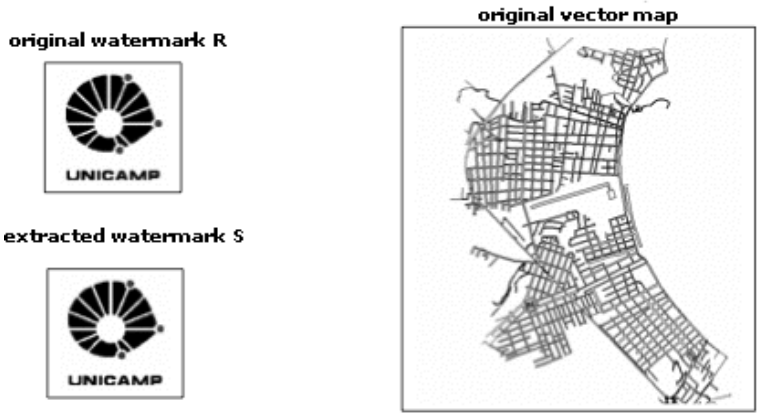


Figure 8. Simple test with no attacks. $r = 1$ $h = 5$

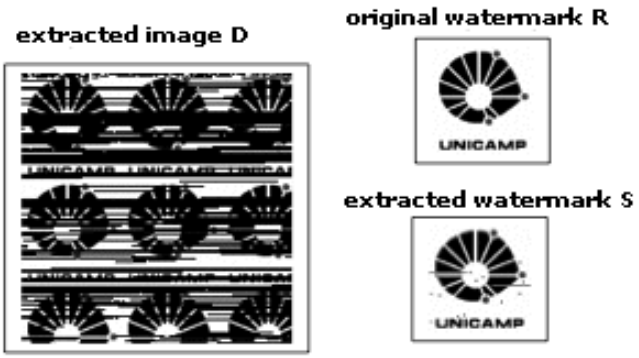


Figure 9. Test with a combined attack $r = 0.99$ $h = 5$

efficiency of the method. On the other hand, the detection algorithm would be simpler, which may facilitate the watermark extraction and, therefore, decrease its effectiveness.

References

Boissonnat, J.-D. and Teillaud, M. (1993). On the randomized construction of the delaunay tree. *Theor. Comput. Sci.*, 112(2):339–354.

Gou, H. and Wu, M. (2004). Data hiding in curves for collusion-resistant digital fingerprinting. In *ICIP*, pages 51–54.

Marques, D. A. (2005). Marcas d'Água visuais em mapas vetoriais. Master's thesis, UNICAMP.

Ohbuchi, R. and Masuda, H. (2000). Managing cad data as a multimedia data type using digital watermarking. In *Knowledge Intensive CAD*, pages 103–116.

Ohbuchi, R., Ueda, H., and Endoh, S. (2003a). Robust watermarking of vector digital maps. In *Proc. IEEE Conference on Multimedia and Expo 2002*.

Ohbuchi, R., Ueda, H., and Endoh, S. (2003b). Watermarking 2d vector maps in the mesh-spectral domain. In *SMI '03: Proceedings of the Shape Modeling International 2003*, page 216, Washington, DC, USA. IEEE Computer Society.

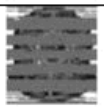
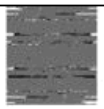
removed points	extracted watermark	r	h	removed points	extracted watermark	r	h
0 %		1,000	5	85 %		0,808	5
40 %		0,999	5	90 %		0,709	4
60 %		0,978	5	95 %		0,561	2
80 %		0,893	5	98 %		0,280	0

Figure 10. Results for several cropping attacks

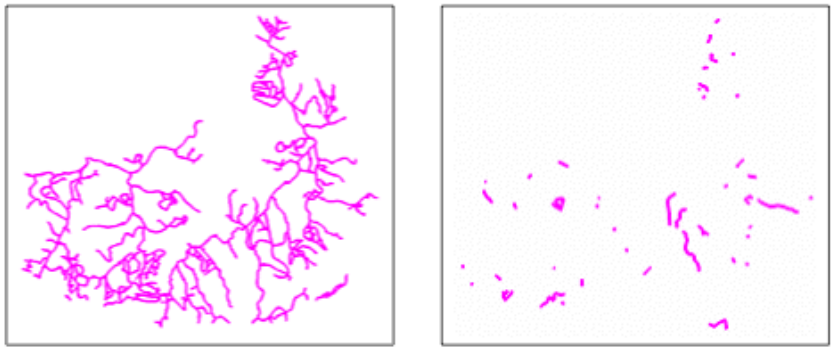


Figure 11. The vector map after a 90% cropping attack

Praun, E., Hoppe, H., and Finkelstein, A. (1999). Robust mesh watermarking. In Rockwood, A., editor, *Siggraph 1999, Computer Graphics Proceedings*, pages 49–56, Los Angeles. Addison Wesley Longman.

Shao, C. Y., Wang, H. L., Niu, X. M., and Wang, X. T. (2005). A shape-preserving method for watermarking 2d vector maps based on statistic detection. *IEICE Transactions on Information and Systems*, E89-D:1290–1293.

Sion, R. (2002). Power: A metric for evaluating watermarking algorithms. *itcc*, 00:0095.

Sonnet, H., Isenberg, T., Dittmann, J., and Strothotte, T. (2003). Illustration watermarks for vector graphics. In *PG '03: Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, page 73, Washington, DC, USA. IEEE Computer Society.

van Wamelen, P., Li, Z., and Iyengar, S. (1999). A fast expected time algorithm for the point pattern matching problem. Technical report, Louisiana State University, Dept. of Mathematics.

Voigt, M. and Busch, C. (2002). Watermarking 2d-vector data for geographical information systems. In *Proc. SPIE, Security and watermarking of Multimedia Content*, pages 621–628.

Voigt, M., Yang, B., and Busch, C. (2004). Reversible watermarking of 2d-vector data. In *MM&Sec '04: Proceedings of the 2004 workshop on Multimedia and security*, pages 160–165, New York, NY, USA. ACM Press.