Illustration Watermarks for Vector Graphics

Henry Sonnet Tobias Isenberg Jana Dittmann Thomas Strothotte

Department of Simulation and Graphics
Otto-von-Guericke University of Magdeburg
{sonnet@isg|isenberg@isg|jana.dittmann@iti|tstr@isg}.cs.uni-magdeburg.de

Abstract

Digital watermarking is a technique for embedding information into data, such as images, 3D models, or audio files, such that some properties (i. e., security, imperceptibility, robustness) are maintained. While most of the existing watermarking techniques focus on encoding copyright information where security is one of the most important properties, we have developed algorithms for embedding Illustration Watermarks, i. e., content-related annotations for the image. Robustness against common media transformations, high capacity, and blind detection are our aspired properties while security and the usage of secure keys are less important. The medium that we are using for embedding data are 2D vector graphics. We introduce algorithms that change line attributes, introduce new vertices in certain patterns, and replace existing stroke segments by new lines in a stylistic way. Based on the modification they introduce, we categorize our techniques into whether they change the appearance of the image or not and whether the changes are perceivable by the naked eye or not. We demonstrate our techniques with silhouette lines obtained from 3D models. Such line drawings are a very common style utilized in many illustrations, in particular in the medical and technical domain.

1. Introduction

Digital watermarking techniques based on steganographic systems offer the possibility of embedding information directly into the media data (see, for example, COX et al. [4] or DITTMANN [5]). Digital watermarking represents an efficient technology to ensure both data integrity and data origin authenticity known as copyright or authentication watermarking. Watermarking techniques are usually used for digital imagery while audio and 3D-models are relatively new application domains. Besides embedding copyright, customer, or integrity information as transparent patterns using a secret key, digital annotation watermarking becomes popular to integrate additional data into the media itself, e. g., in smart images as shown by ALATTAR [1]. The most important properties of digital watermarking techniques are robustness, security, imperceptibility/transparency, complexity, capacity, and possibility of verification. Although a wide variety of techniques have been proposed, there are no accepted standards as to their classification nor their quality measurement. Depending on the application these parameters differ and cannot be optimized separately. For our illustration scenario, high transparency of the embedded information, robustness against common media transformations, high capacity (payload), and blind detection (watermark retrieval without original) are key issues while security of the watermark and the usage of secure keys are less important. Therefore, our goal is to design an appropriate *Illustration Watermark* for 2D vector data.

Line drawings as representatives of 2D vector data are a very common style used in illustrations [6], particularly in the medical and technical domain. In recent years, a wide variety of techniques have been proposed for the automatic generation of, for example, silhouette line drawings from three-dimensional geometric models [8, 18]. Many of these algorithms produce the silhouette lines in analytic form, i. e., these lines are not hidden in a pixel image and can be used for further processing. Usually, line styles are applied to the silhouettes used in interactive applications [9, 10]. However, since the silhouettes are available in an analytic form, they can also be used to create vector graphics.

The advantages of vector graphics in terms of typically small storage costs and high reproduction quality are widely known not only with respect to print but also on-screen. However, when vector graphic illustrations are processed (import, export, and manipulation in vector graphics packages such as COREL DRAW), potentially attached information in form of text files about what is depicted will most certainly get lost. Thus, it is of great benefit if this additional information can be embedded within the graphic itself. The embedded information has to be stored in such a way that it does not get lost when applying common geometrical transformations. For example, typical transformations

used in vector graphics processing such as translation, rotation, scaling, and even partial zooming must not destroy the information embedded within the graphic. The additional information should also not get lost when changing from one standard format to another (e. g., PDF to POSTSCRIPT).

We demonstrate the suggested methods for *Illustration Watermarking* for vector graphics by applying them to computer-generated silhouette line drawings. However, they can be used with any vector graphic based on lines. Our main goal is to embed information without actually altering the visual appearance of the graphic or at least without changing it perceivably. We also introduce a method which does change the visual appearance of the graphic perceivably but does it in a stylistic way.

As shown in Figure 1, our techniques are based on the following procedure. We start with a 3D geometric model and first compute its visible silhouette. The visible silhouette lines serve as the input for the *Illustration Watermarking* algorithms. The information is embedded by adding or altering the positions of vertices within a stroke or by changing the line attributes at certain vertex positions. The resulting data is output into a PDF file in order to allow further processing of the vector graphic (e. g., transformations into other vector graphic formats).

In the remainder of this paper, we first analyze related work in Section 2. Afterwards, we introduce the term *Illustration Watermark* and discuss what this special kind of watermarks might be used for in Section 3. A detailed description of our introduced algorithms and their categorization is presented in Section 4. An evaluation of our algorithms regarding their amount of encodable data, perceivability, and robustness is carried out in Section 5. Section 6 comprises conclusions to this paper.

2. Related Work

There are a wide variety of watermarking techniques in the domains of audio, video, and image processing [4, 5, 3, 15, 19]. The number of watermarking techniques for 3D geometric models has been continuously increasing since about 1997. The techniques introduced in this paper benefit to a great extent from these methods. Hence, we survey a selection of 3D and 2D approaches in the object space which are mainly used for copyright issues without giving a complete overview which would be beyond the scope of this paper.

OHBUCHI et al. were the first to embed digital watermarks into 3D geometric models [12]. The first algorithm they describe searches sequentially for four adjacent triangles sharing edges. The data is embedded by changing the ratio of specific edges and heights. Other algorithms vary the ratio of volumes of a pair of tetrahedrons or peel off triangle strips from a given triangle mesh, whose adjacency embed binary data. In one of their latest publications, OHBUCHI et al. modify the amplitude of mesh spectral coefficients and hence change mesh shapes in their transformed domain [13]. The first digital watermarking technique for 3D polygonal meshes using a domain transformation (based on wavelet transform and multiresolution representation) was introduced by KANAI et al. [11]. The embedded watermark is imperceptible and invariant to affine transformations. In 1999, BENEDENS described an algorithm whose primary aim was to be robust against point randomization, mesh altering operations, and polygon simplification [2]. In order to embed binary data, he computed groups of similar normals of the model in advance (so-called *bins*) and moves them slightly.

A watermarking technique based on 2D vector data for Geographical Information Systems (GIS) is presented by VOIGT and BUSCH [20]. They add pseudo-noise (*PN*) to the obtained coordinates within a certain tolerance of the data. Their method is robust against attackers as long as they change the coordinates within the tolerance range. In order to detect those attacks and the embedded information, the PN-sequence has to be known by the embedder and the decoder. OHBUCHI et al. also address the subject of watermarking 2D vector maps in the geographical data domain [14]. After creating a single mesh of the collection of polygons and polylines using delaunay triangulation, they compute multiple watermarking patches. In order to embed watermarking signals, the calculation of mesh spectral coefficients and their modification is necessary. As a result, they adopted their aforementioned approach ([13]) for geometrical 3D models and applied it to 2D vector data. Their procedure shows that there is some similarity in watermarking between 3D and 2D data in object space. Thus, methods based on 3D data are easier to apply to 2D vector data than to raster graphics. SOLACHIDIS et al. introduce another approach that utilizes a domain transformation [17]. They first combine the coordinates of a polygonal line to represent the computed signals by their Fourier descriptors (*FD*) before they change the magnitude of the *FD* to embed watermarking signals. In the broader course, the invariance of their algorithm to a number of geometric manipulations (translation, rotation, scaling, change of starting point, and inversion of traversal direction) is shown. Very closely related is also the approach by SCHMUCKER who embeds watermarks in music scores by modifying features of the notation [16] or the work by HUANG and YAN who alter the spacing of texts in order to embed additional information [7].

3. Illustration Watermarks

We define *Illustration Watermarks* as a representation of information within an image which enables end-user interaction with the image. In this section, we will discuss the purpose of such watermarks and their necessity, while the

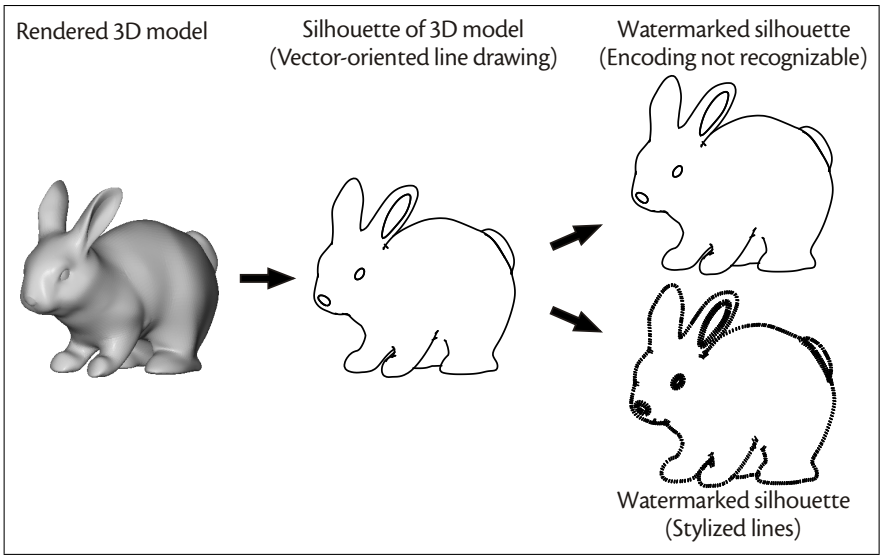

Figure 1: Starting from a 3D model, the silhouette is computed and used to embed data with or without recognizable changes of the graphic. Other sources for 2D vector graphics are possible as well.

remainder of the paper will then focus on algorithms for creating such illustration watermarks.

Interaction with an image is to serve one or more of several purposes:

- *Image Enrichment*: A single image may not contain the information which an end-user needs for a particular context. At the same time, the author of the image may have additional information at their disposal which could augment it. Such information is to be included in the image as a watermark.

- *Context Dependent Visualization*: A fixed image cannot change its appearance depending on the context in which it is being used. However, there are situations in which slight changes may be foreseen by the author of the image or the end-user, depending on how the image is to be used. Such changes in the visualization of the underlying image data can be included as an illustration watermark and selectively extracted to tune the image to the context of its use.

Examples of image enrichment include information about what is happening in the image what certain objects may be used for. For example, if the image contains an image of a heart, it may be useful to provide information about the direction of the flow of blood should the user want it. Such information can be encoded within the image as a watermark and extracted by the user on demand. The information could then be visualized as arrows temporarily placed within the image.

An example of a context dependent visualization is the rendition style to be used. For example, an engineering drawing generally contains information as to the sizes of objects, while a rendition of the same object intended for end-user documentation might typically be required in a photo-realistic manner. Alternative visualizations of one and the same object can be encoded within an image as a watermark and activated by the user as needed.

Such additional information about an image contained within an illustration watermark can be extracted by the user with an appropriate browser. Each image has its "normal" appearance which is visualized with any browser supporting the image's format. The watermarks now provide the end-user with the ability to extract additional information which (hopefully) will be useful in their dialogue context. A specially equipped browser will enable the end-user to inspect what watermarks are present and to see their effect.

This methodology raises a number of questions. Perhaps the most compelling is why the information needed for the dialogue should be encoded as a watermark within the image rather than simply being supplied with an additional data structure. The main reason for exploring watermarks in this context is that an encoding as an image enables users to work with all the information pertaining to it in a uniform and systematic manner. It enables end-users to use standard tools for image manipulation (like file format conversion, and cut-and-paste operations) rather than having to deal with a separate tool for the additional information. It ensures the consistency of the image and its additional information which was intended by the image author, as it is not possible to inadvertently alter or remove the additional information. The technology for robustness and security developed for watermarks in general apply to illustration watermarks, also.

Of course, the use of illustration watermarks also has its disadvantages. Perhaps the most important is the limited amount of information which can be encoded within a watermark. However, as we shall see in the remainder of this paper, indeed that amount is substantial (at least a few thousand bytes), which is sufficient for many applications. It may also be that the amount of memory needed to encode the additional information as a watermark is greater than as a simple and separate data structure. However, the overall amount of memory needed is also not really an issue anymore today in many applications. Hence there are a variety of situations in which the advantages far outweigh the disadvantages, making this a viable technology to investigate.

In this paper we focus on illustration watermarks for vector graphics. Certainly, vector graphics which lead to line-drawings as a style of non-photorealistic rendering is a class of graphics in which illustration watermarks are of particular value. An illustration consisting of a line drawing as its basis will often be augmented by various different pieces of information, depending on the context of the end-user requirements. Hence, vector graphics are an important class of graphics for which illustration watermarks will form an important basis for encoding information which should not appear within the illustration all at once.

The end-user issues of illustration watermarking are beyond the scope of this paper and will be dealt with by the authors elsewhere. The remainder of this paper focuses on algorithms for encoding information as illustration watermarks in vector graphics.

4. Implementation Details

In this section, we propose four different algorithms designed for embedding binary data in line graphics. We classify these algorithms according to whether they change the appearance of the image or not. Algorithms which do not change the appearance of images only enlarge the graphic by introducing additional points. In contrast, algorithms which change the appearance of images work by modifying line properties such that they may change the perception of the graphic or replace existing line strokes by new lines. For the algorithms which do change the appearance, we also make the distinction whether this modification is perceivable or not. The binary data that is to be embedded within the vector graphic can be represented as bi-level pictures as well as character strings.[1] We use a bit stream generated from this input data to embed it in the image.

Besides the data to be embedded, we use a vector-oriented line drawing as input for each algorithm. As shown in Figure 1, we interactively generate such a line drawing by rendering a 3D model, moving the camera to get a suitable view onto the model, and finally compute the silhouette of the model (see e. g., [8]). The silhouette (a polyline) is composed of a number of line segments that can be stored in a vector-oriented file format, in our case the (ASCII based) PDF file format.

[1] Of course, other types of data may be used. We chose to restrict ourselfs to use ASCII text and dither images for demonstration purposes, only.

4.1. Algorithms That Do Not Change the Images' Appearance

The first group of techniques illustrates two algorithms which modify the original line graphic in such a way that is not perceivable to the viewer. The embedding of binary data into the image only causes an enlargement of the given graphic data in terms of storage costs.

4.1.1. Additional Points Per Segment

The first algorithm is based on inserting additional points into the image. Figure 2 illustrates the main steps:

- **Subdivision.** Each line segment of the given original polyline (the silhouette) is subdivided such that none of the computed line segments is longer than a specific, user definable length. This is an optional step which increases the number of segments that can be used for embedding data.

- **Insertion.** After the two vertices of each new line segment, an additional vertex is inserted. However, the new vertex is located between those two vertices. The distance to the first of the two vertices mirrors the encoded binary data.

The illustration of Figure 2(b) shows the subdivision step after which each line segment L_j ($j = \{1, ..., M\}$ with $M =$ number of line segments) consists of N_{L_j} vertices; possible vertex positions for encoding binary data can be seen in the enlargement of Figure 2(b) (only one of the positions is used). Accordingly, $\tilde{P}_{i,j}$ is the inserted vertex between $P_{i,j}$ and $P_{i+1,j}$ ($i = \{1, ..., N_{L_j}\}$). For example, if $\tilde{P}_{i,j}$ is equal to $P_{i,j}$, bit 0 is encoded; if $\tilde{P}_{i,j}$ is in the middle between $P_{i,j}$ and $P_{i+1,j}$ it encodes bit 4. Since we only code bits which equal 1, we have to specify when the next byte starts. We do so by adding a vertex $P_{i+1,j}$ at the end of the list in order to signal the end of the current byte. In general, the position of the inserted vertex can be computed by the following equation based on the binary data to be encoded:

$$\tilde{P}_{i,j} = P_{i,j} + \frac{bit}{8} \cdot (P_{i+1,j} - P_{i,j}) \quad (0 \leq bit \leq 8)$$

bit ranges from 0 to 7 specifying the according bit and is equal to 8 if the start of the next byte should be indicated.

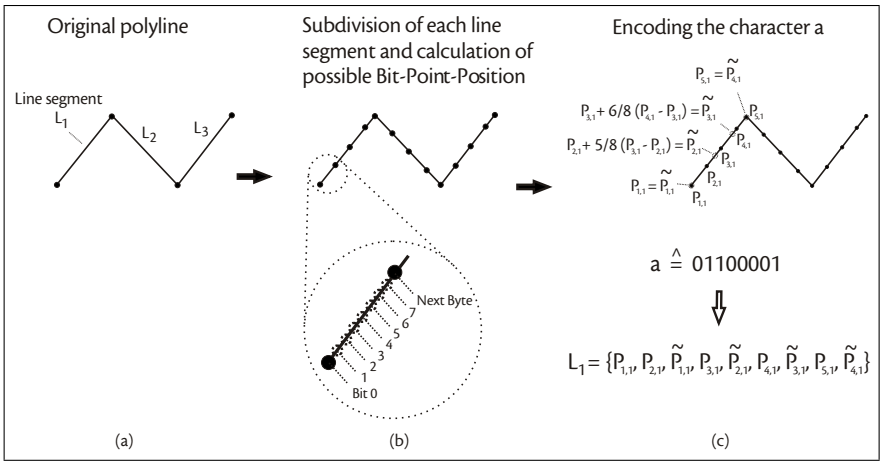

Figure 2: A given polyline is subdivided before the line segment points are used to insert additional points responsible for embedded binary data. The modification is not visible to the viewer.

As an example, the character a is encoded into the line segment L_1 in the illustration of Figure 2(c). The associated binary code (01100001) yields the new point field:

$$L_1 = \{P_{1,1}, P_{2,1}, \tilde{P}_{1,1}, P_{3,1}, \tilde{P}_{2,1}, P_{4,1}, \tilde{P}_{3,1}, P_{5,1}, \tilde{P}_{4,1}\}$$

The watermark detection and retrieval is designed as a blind watermarking scheme. This means that for decoding the embedded information neither the original line drawing nor encoding details regarding any point distances are necessary. The line segment points are only traversed and analyzed if they are positioned between the two points belonging to the subdivided line segment and are located to the left in the point field. If they are between them or equal to one of them, the distance to the first of the two points is calculated and converted to binary data. This means that the coefficient c ($c = bit/8$) has to be computed before the encoded bit can be extracted:

$$c = \frac{\tilde{P} - P_{i,j}}{P_{i+1,j} - P_{i,j}} \quad \text{(for both components (x,y))}$$

$$bit = floor(c \cdot 8 + 0.5) \quad \text{(floor: greatest integral value)}$$

Figure 3(b) shows an example. The enlargement demonstrates that there is no difference detectable compared to the original part in Figure 3(a).

One modification of the algorithm which does not require drawing back on the original segment would simply switch the last two points of each segment. I. e., instead of drawing two points of original segment followed by the additional vertex, the first original vertex is drawn followed by the vertex carrying the information which is followed by the second original vertex. This way, a stroke consisting of a series of segments would not need to be split into several small strokes each consisting of only two segments. Instead, the long stroke would be preserved saving storage costs.

4.1.2. Line Segment Length

The second algorithm that does not modify the appearance encodes binary data by varying the length of the line segments. Thereby, we use a different line subdivision scheme than before in order to embed the binary data into the lengths of the subdivided line segments. Given a specific base length l which is typically at least about a magnitude smaller than the length of the smallest line segment, the subdivision of the original line segments is performed as follows:

1. Determine the length l_{L_j} of the current original line segment and add its first point to the point field F_P. F_P is a field which contains all points which result from line subdivision. The current length of the subdivided line segment l_{cur} is 0.

2. The new length $\tilde{l}$ is calculated according to the bit position to be encoded (again, we only encode the bits that are 1):

$$\tilde{l} = \frac{1}{8} \cdot (bit + 1) \cdot l \qquad (0 \leq bit \leq 7)$$

$$\tilde{l} = l + \frac{1}{8} \cdot l \qquad \text{(next byte)}$$

If this length $\tilde{l}$ plus the current segment's length l_{cur} is less or equal to l_{L_j}, we add a new point to F_P located on the original line segment with the distance $\tilde{l}$ to the last point of F_P. Then we add $\tilde{l}$ to l_{cur}.

If $l_{cur} + \tilde{l}$ happens to be larger than the total length of the segment l_{L_j}, no more data can be embedded into this segment. However, just adding the last point of the segment to the list of points would possibly encode a bit which was not intended. In order to avoid this,

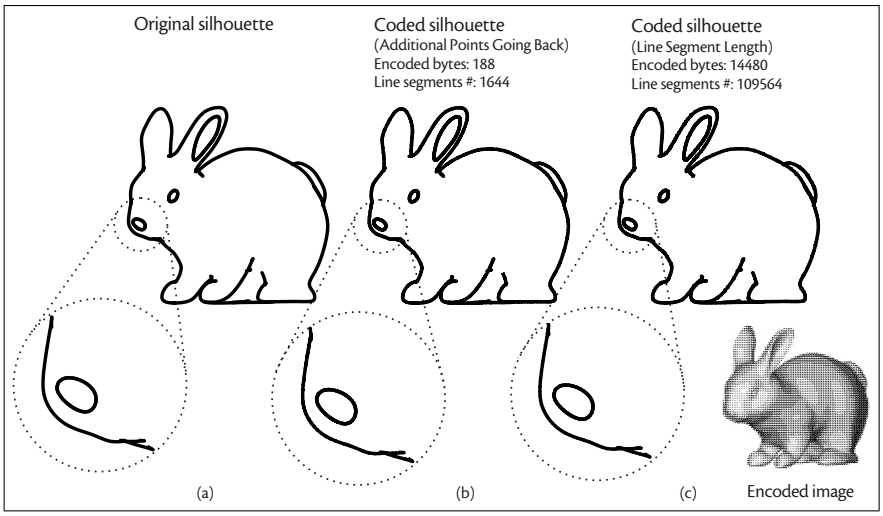

Figure 3: The bunny in (b) was coded by the algorithm that adds points between the endpoints of a line segment; the graphic in (c) was coded by adapting the line segment lengths. During subdivision, more points than in (b) were added with smaller distances such that more data could be embedded, enough to encode the dithered image in the corner right (255x247 pixels).

we continuously add points to F_P which are $\frac{1}{16}l$ apart while $l_{cur} < l_{L_j} - \frac{1}{8}l$.

3. Finally the last point of the original line segment is added to F_P and steps 1 to 3 are repeated for the next line segments.

This approach allows encoding binary data into each computed line segment. Thereby, an amount of data comparable to the algorithm of Section 4.1.1 can be embedded. But the line graphic has not to be enlarged that much because no additional points between the line segments are inserted. The graphic of Figure 3(c) is coded using the algorithm described. It contains an image of size 255 x 247 with a depth of one bit. Encoding an image of this size requires at least 7874 bytes without any compression. The high number of line segments in the image shown enables such an encoding.

When decoding the embedded information, the specific length l has to be known which might be seen as a disadvantage. Therefore, we modified the subdivision step in such a way that the length l is encoded in the first line segment (being the length of this segment). The lengths of the following segments correspond to this specified length. When computing the length $\tilde{l}$ of each remaining line segment the encoded bit can simply be decoded by the following equation:

$$bit = floor\left(\frac{\tilde{l} \cdot 8}{l} + 0.5\right) - 1$$

4.2. Algorithms That Change the Images' Appearance

While the previously discussed algorithms solely enlarge the geometric data without any influence on the appearance of the image, the following algorithms may change the view of the line drawing.

4.2.1. Angled Lines

In this section, we describe an algorithm which changes the appearance of the line drawing significantly but does so in order to create a stylistic appearance for the rendition.

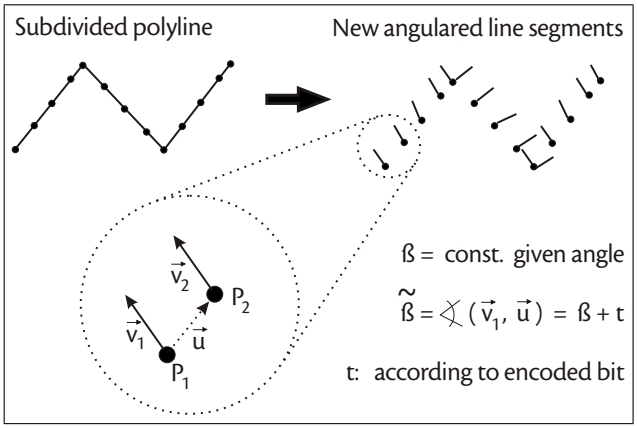

Figure 4: The subdivided line segments are replaced by new, stylized line segments pointing away from the original direction and which carry the embedded data.

Similar to the previous approaches, the algorithm starts by subdividing the original line segments. Nevertheless, now the points of the subdivided line segments are utilized as starting points of new line segments pointing away from the original stroke. The binary data is encoded into the angle $\tilde{\beta}$ between the newly added line segments and the original, not divided line segment. Figure 4 illustrates this pro-

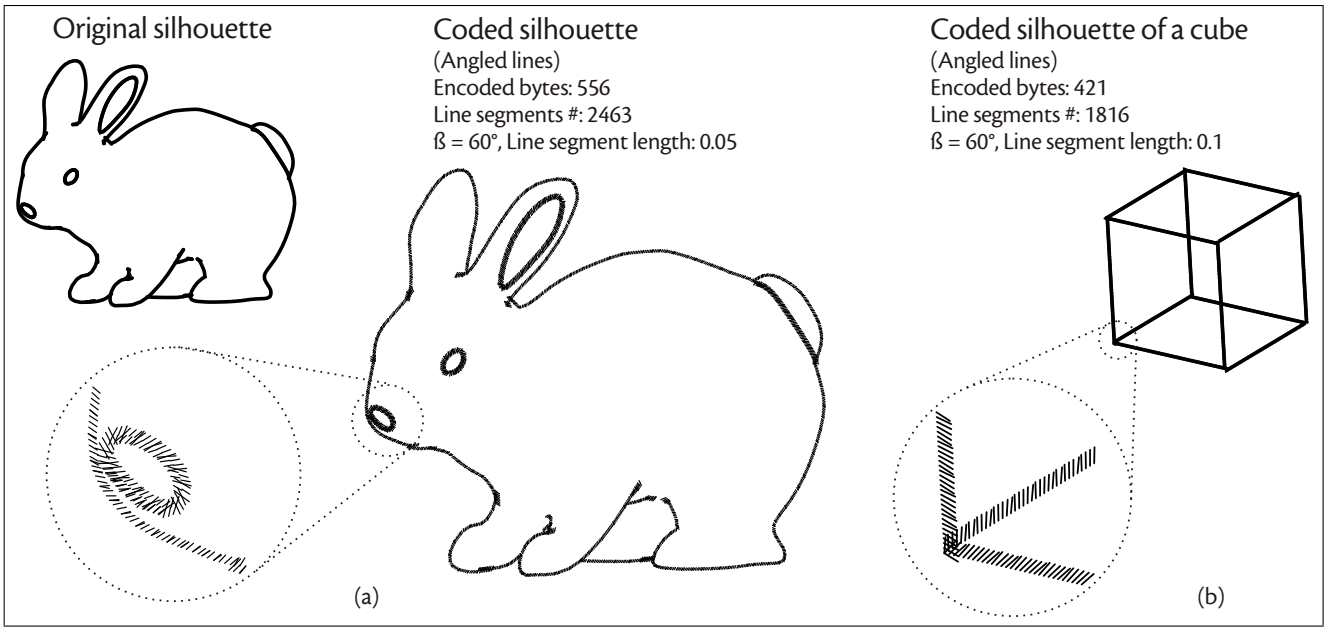

Figure 5: The original line segments are replaced by new lines pointing in another direction. The viewer gets the impression of a line style.

cedure. Given an angle β, the new angle $\tilde{\beta}$ is calculated by adding a bit-specific amount to β. Having determined the angle $\tilde{\beta}$ and the vector $\vec{u}$ between P_{i+1} and P_i, the vector of the new line segment (e. g., $\vec{v_1}$ in Figure 4) can be computed as follows:

$$\begin{pmatrix} v_{i,j_x} \\ v_{i,j_y} \end{pmatrix} = \begin{pmatrix} \cos\tilde{\beta} \cdot \vec{u_{j_x}} & - & \sin\tilde{\beta} \cdot \vec{u_{j_y}} \\ \sin\tilde{\beta} \cdot \vec{u_{j_x}} & + & \cos\tilde{\beta} \cdot \vec{u_{j_y}} \end{pmatrix}$$

Instead of using the original or subdivided line segments, new line segments as a result of the computed vectors $v_{i,j}$ with a user-definable length are rendered (see Figure 5). Besides the length, the distance between those line segments (synonymous with the interval during subdivision) and the angle β can be diversified to get numerous variations among the line drawings. In order to decode the embedded information, β has to be known. Thus, it is to be specified by the user or encoded as the first used angle. The vectors $\vec{u_j}$ are computed as the difference between subsequent starting points of the angled line segments. This way, the angles between the $\vec{u_j}$ and the $v_{i,j}$ can simply be calculated and be used to determine the encoded binary data.

4.2.2. Changing Line Attributes

Another approach for embedding binary data in line graphics is to change some of the line attributes, such as color and width (see Figure 6). These attributes can be modified before each of the line segments is characterized.

For example, in a gray-level image the gray value $\tilde{G}$ can be computed depending on the bit to be encoded as follows:

$$\tilde{G} = 0.1 \cdot 1/8 \cdot bit + G - 0.05 \quad (0 \leq G \leq 0.99)$$

or with somewhat more contrast as seen in the image of Figure 6(a):

$$\tilde{G} = 0.5 \cdot 1/8 \cdot bit + G - 0.2 \quad (G = 0.5)$$

G is the base gray value; the modified gray values are computed so that they are close to G. If G is not modified this again signals the start of a new byte. Care has to be taken in adjusting the two parameters when G is set to be 0 (black) or 1 (white) in order to avoid gray values greater than 1 or less than 0. According to the range within the computed gray values, the modification of the graphic is more or less visible to the viewer.

It is relatively easy to manipulate these attributes and thus the embedded data without changing the geometry data at all. Indeed, this prompted OHBUCHI et al. to conjecture that the geometry of objects is the best candidate for data embedding, since it is the least likely to be removed [12]. However, the widths as well as the color values can be modified marginally such that no data encoding is detectable at first sight.

5. Evaluation and Discussion

We were interested in how to appraise our presented algorithms especially regarding a number of aspects. We examined how much data can be embedded and what additional storage space is needed. Second, we analyzed the algorithms as to whether there are any differences between the coded graphic and the original one. In addition, we examined our techniques regarding robustness against trans-

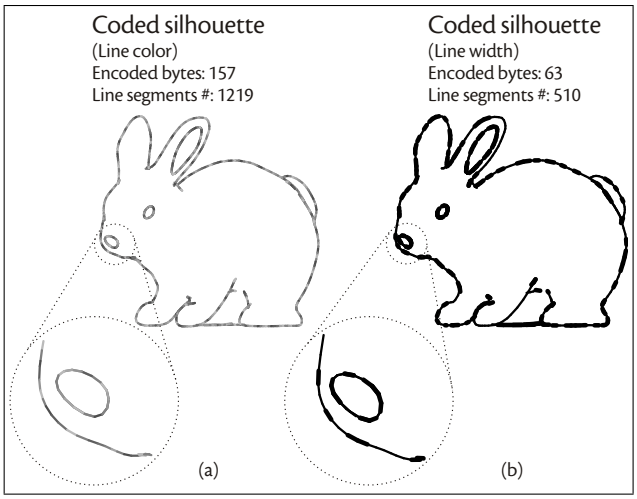

Figure 6: Line attributes are used to embed binary data. In (a), the color attribute is modified whereas in (b) the line width was diversified to encode information. In both graphics the attribute variations are overstated in order to illustrate them in a better way.

formations such as translation, rotation, and scaling. In particular, we analyzed how import and export operations influence the decoding process and whether the encoded information is at least partially accessible after removing parts of the graphic.

5.1. Amount of Encodable Information

The amount of data that can be encoded depends on the number of line segments that can be increased by the subdivision step (see Section 4.1.1). The smaller the distance between two consecutive line segment points, the more binary data can be embedded. Certainly, this causes an insertion of additional points and consequently an increase in storage requisition.

Table 1 shows measurements for each algorithm.[2] The number of line segments of the original line drawing and the appropriate number after subdivision are shown as well as the number of encoded data and the resulting size of the PDF file. It is evident that the number of bytes encoded and at the same time the size of the PDF file increases when the distance between consecutive points is abbreviated resulting in more line segments. The largest number of bytes are embedded using the *Line Length* algorithm. The reason is that during the subdivision more line segments, compared to the other algorithms, are generated due to some shorter segments holding the lower bit positions. The diagram of Figure 7 illustrates that the *Line Length* algorithm is also the most efficient while the *Line Color/Width* algorithm yields the largest PDF files. Further more, a nearly linear correlation between the embedded amount of data and the resulting data size of the PDF file is recognizable. It can also be observed that there is a disproportion of the *Line Length* algorithm at the beginning. Indeed, the *Line Length* algorithm yields many more line segments but less encoded bytes compared to the other algorithms. The explanation for this behavior is the insertion of very short line segments. These line segments are inserted as long as the current bit-specific line length exceeds the length of the original line segment (recall Section 4.1.2).

[2] Please note that we used ASCII-based PDF files. Using binary coding will certainly improve the storage efficiency of Illustration Watermarks.

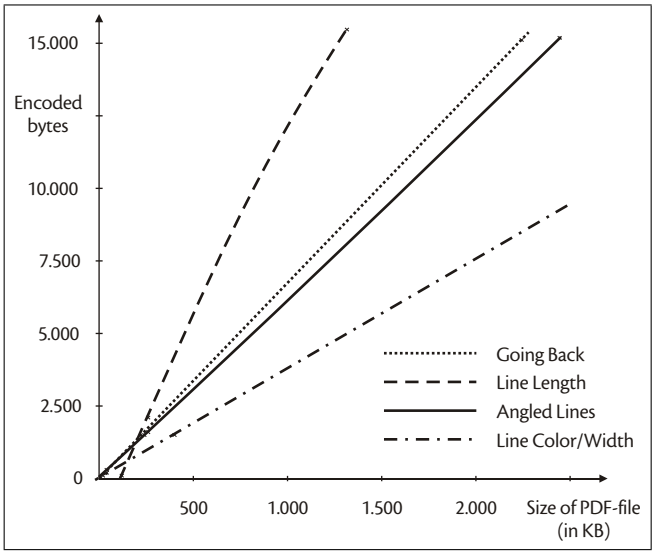

Figure 7: Memory requirements for embedding an amount of data (in bytes) for each of the presented algorithms.

5.2. Recognizable Differences

When we presented the algorithmic details, we categorized the Illustration Watermark algorithms into changing the appearance of the image or not according to the modifications they introduce. The only algorithm that makes significant changes to the appearance of the image is the *Angled Lines* algorithm. When parameterized accordingly, the diversification of line color and width does not cause recognizable effects. In Figure 6, we have chosen two extreme examples in order to better illustrate the techniques. Nevertheless, it is usually sufficient to vary the color and width minimally such that it is not detectable when viewing the image with the naked eye (which is a subjective test, of course). However, when decoding the data algorithmically, even small changes in attribute values are enough to be recognizable.

The modifications resulting from the *Going Back* and *Line Length* algorithms are not perceivable by the viewer. However, there is a problem that might occur due to numerical roundoff errors. If the distance between the subsequent points on a line segment during the subdivision is chosen

Algorithm	*# Lines (not subdivided)*	*# Lines (subdivided)*	*Encoded bytes*	*Size of coded ASCII PDF file*	*Ratio of data hidden per storage cost (%)*
Going Back	71	1226	157	35.0 KB	0.44
	76	7135	1490	236.9 KB	0.61
	79	67358	15113	2.2 MB	0.66
	68	633878	143255	21.1 MB	0.65
Line Length	71	7437	134	129.1 KB	0.10
	76	15537	2127	267.5 KB	0.78
	79	104010	22200	1.7 MB	1.25
	68	938895	211037	15.7 MB	1.28
Angled Lines	71	1226	276	45.4 KB	0.59
	76	7135	1614	259.2 KB	0.61
	79	67358	15236	2.4 MB	0.61
	68	633878	143376	22.4 MB	0.61
Line Color/ Line Width	71	1226	157	42.7 KB	0.36
	76	7135	1490	394.1 KB	0.37
	79	67358	15113	3.9 MB	0.37
	68	633878	143255	36.8 MB	0.37

Table 1: Comparison of our algorithms regarding the number of line segments before and after subdivision, the amount of encoded bytes and the required storage capacity.

too small (in our tests less than 0.001 units), the viewer might detect some tiny interruptions on a printed copy of the graphic depending on resolution and printer type.

The procedure of reconstructing the original vector graphic from one with embedded data is very much straightforward. When the *Line Color/Width* algorithm was used, the only thing to do is to remove the attributes which carry the information. For the remaining algorithms, the decoding techniques have to be applied which makes it possible to reconstruct the start and end points of the original line. However, an additional line subdivision which could have been applied in order to store more data cannot easily be removed other than searching for consecutive straight line segments and merging those.

5.3. Robustness

To examine the robustness of our algorithms we carried out a number of subjective tests. In order to verifying the robustness against geometrical transformations, we imported the PDF file into CORELDRAW, applied a number of common transformations (translation, scaling, rotation), exported the data as a EPS file, and transformed this file back to PDF. Additional file conversions were done by opening the PDF file in GSVIEW, converting it to another vector format (POSTSCRIPT file format), afterwards to the EPS file format and in the end back to the PDF file format using `epstopdf` under a UNIX operating system.

The information encoded by the *Angled Lines* and *Line Color/Width* algorithms could be decoded after the aforementioned transformations and conversions without any problems. The information encoding done by the two other algorithms (*Going Back* and *Line Length*) caused some decoding problems after the robustness tests. The first problem was that the last of two identical consecutive points was removed by CORELDRAW's export function when saving the modified file. Recall, the *Going Back* algorithm where the start of the next byte was signaled by inserting a point which should be located between two of the previous points (in the case of the start of the next byte it should be identical with the second one, see Section 4.1.1). As a result of the removal of points, the starting of the next byte could not be identified. To avoid this problem, we multiplied the point to be inserted with a factor of 0.9999 in order to get a nearly exact copy of the previous point. Because of numerical roundoff errors and minutely different scaled point coordinates caused by internal system conversions (appearing in CORELDRAW) as the second problem of both algorithms, a small error tolerance was incorporated in the decoding process. Introducing error tolerances solved both the problem with the modification of points to avoid two subsequent identical points as well as the problem caused by internal system conversions.

After removing parts of the graphic most of the information embedded in the remaining graphic was still decodable since the beginning of each new byte is indicated. Therefore, none of the bits encoded were associated with an incorrect byte. However, our major goal was to embed a large amount of connected data. If parts of the graphic were removed the embedded data is incomplete and important information

might get lost. In addition, we optimized the algorithms in order to provide higher capacity and robustness rather than security. An observer who knows the applied algorithms could easily figure out that there is information hidden in the lines. However, the indended application does not require high security.

6. Conclusion

In this paper, we examined ways to embed Illustration Watermarks into vector graphics. Illustration Watermarks were defined as additional, content-related information which is added to an illustration. Instead of storing the information separate from the image which carries the illustration (where it could get lost through transformations on the image) we embed the data into the image itself.

Illustrations are often created as line drawings. These, in turn, can be represented as vector graphics which has numerous advantages in terms of reproduction quality and storage efficiency. We demonstrate how to embed Illustration Watermarks in those vector graphics using four different approaches. We classify these approaches according to their ability to preserve the image's appearance. The *Going Back* and *Line Length* algorithms do not introduce any perceivable changes whereas the *Line Color/Width* algorithm can be parameterized so that the changes are not visible to the naked eye. In contrast, the *Angled Lines* algorithm exchanges the stroke segments by stylized lines which carry the data. All our approaches store the embedded information locally, i. e., this allows us to add annotiations specifically to those parts to which they belong. In addition, when parts of the graphic are removed, the information of the remaining parts can still be recovered.

In all algorithms, we used the bit positions for encoding. Other schemes for storing binary data as well as efficient data compression can be used with only little changes to the algorithms. In particular, the latter would greatly improve the data efficiency of our algorithms.

We showed that the data embedding is robust against common transformations and format conversions which are often applied to vector graphics.

References

[1] A. M. Alattar. Smart Images Using Digimarc's Watermarking Technology. In *Proc. SPIE Security and Watermarking of Multimedia Contents II*, pages 264–273, 2000.

[2] O. Benedens. Geometry-Based Watermarking of 3D Models. *IEEE Computer Graphics and Applications*, 19(1):46–55, 1999.

[3] B. Chen and G. W. Wornell. Quantization Index Modulation: A Class of Provably Good Methods for Digital Watermarking and Information Embedding. *IEEE Transactions on Information Theory*, 47(4):1423–1443, May 2001.

[4] I. J. Cox, M. L. Miller, and J. A. Bloom, editors. *Digital Watermarking*. Morgan Kaufmann, San Francisco, 2001.

[5] J. Dittmann. *Digitale Wasserzeichen*. Springer Verlag, Heidelberg, 2000.

[6] E. R. S. Hodges, editor. *The Guild Handbook of Scientific Illustration*. John Wiley & Sons, 2nd edition, 2003.

[7] D. Huang and H. Yan. Interword Distance Changes Represented by Sine Waves for Watermarking Text Images. In *Proc. Pan-Sydney Area Workshop on Visual Information Processing (VIP 2000)*, 2000.

[8] T. Isenberg, B. Freudenberg, N. Halper, S. Schlechtweg, and T. Strothotte. A Developer's Guide to Silhouette Algorithms for Polygonal Models. *IEEE Computer Graphics and Applications*, 23(4):28–37, July/Aug. 2003.

[9] T. Isenberg, N. Halper, and T. Strothotte. Stylizing Silhouettes at Interactive Rates: From Silhouette Edges to Silhouette Strokes. *Computer Graphics Forum*, 21(3):249–258, Sept. 2002.

[10] R. D. Kalnins, L. Markosian, B. J. Meier, M. A. Kowalski, J. C. Lee, P. L. Davidson, M. Webb, J. F. Hughes, and A. Finkelstein. WYSIWYG NPR: Drawing Strokes Directly on 3D Models. In *Proc. SIGGRAPH 2002*, pages 755–762, Reading, MA, 2002. Addison Wesley.

[11] S. Kanai, H. Date, and T. Kishinami. Digital Watermarking for 3D Polygons using Multiresolution Wavelet Decomposition. In *Proc. 6th IFIP WG 5.2 International Workshop on Geometric Modelling*, pages 296–307, 1998.

[12] R. Ohbuchi, H. Masuda, and M. Aono. Watermarking Three-Dimensional Polygonal Models. In *Proc. ACM Multimedia 1997, 5th ACM International Multimedia Conference*, pages 261–272, 1997.

[13] R. Ohbuchi, A. Mukaiyama, and S. Takahashi. A Frequency-Domain Approach to Watermarking 3D Shapes. *Computer Graphics Forum*, 21(3):373–382, Sept. 2002.

[14] R. Ohbuchi, H. Ueda, and S. Endoh. Watermarking 2D Vector Maps in the Mesh-Spectral Domain. In *Proc. International Conference on Shape Modelling and Applications*, pages 216–225, 2003.

[15] F. A. P. Petitcolas, R. J. Anderson, and M. G. Kuhn. Information Hiding – A Survey. In *Proc. IEEE, Special Issue on Protection of Multimedia Content*, pages 1062–1078, 1999.

[16] M. Schmucker. Using Musical Features for Watermarking Music Scores. In *Proc. 1st Int. Conf. on WEB Delivering of Music (Wedelmusic 2001)*, pages 20–25. IEEE Computer Society, 2001.

[17] V. Solachidis, N. Nikolaidis, and I. Pitas. Watermarking Polygonal Lines Using Fourier Descriptors. In *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'2000)*, pages 1955–1958, 2000.

[18] T. Strothotte and S. Schlechtweg. *Non-Photorealistic Computer Graphics: Modeling, Rendering, and Animation*. Morgan Kaufmann, San Francisco, 2002.

[19] M. D. Swanson, M. Kobayashi, and A. H. Tewfik. Multimedia Data-Embedding and Watermarking Technologies. In *Proc. IEEE (Special Issue on Multimedia Signal Processing)*, volume 86, pages 1064–1087, 1998.

[20] M. Voigt and C. Busch. Watermarking 2D-Vector Data for Geographical Information Systems. In *Proc. IS&T/SPIE Electronic Imaging 2002*, volume 4675, pages 621–628, 2002.