PAPER

Data Hiding in Computer-Generated Stained Glass Images and Its Applications to Information Protection

Shi-Chei HUNG[†*], ***Nonmember***, **Da-Chun WU**[††a)], ***Member***, ***and*** **Wen-Hsiang TSAI**[†††,††††], ***Nonmember***

SUMMARY The two issues of art image creation and data hiding are integrated into one and solved by a single approach in this study. An automatic method for generating a new type of computer art, called stained glass image, which imitates the stained-glass window picture, is proposed. The method is based on the use of a tree structure for region growing to construct the art image. Also proposed is a data hiding method which utilizes a general feature of the tree structure, namely, number of tree nodes, to encode the data to be embedded. The method can be modified for uses in three information protection applications, namely, covert communication, watermarking, and image authentication. Besides the artistic stego-image content which may distract the hacker's attention to the hidden data, data security is also considered by randomizing both the input data and the seed locations for region growing, yielding a stego-image which is robust against the hacker's attacks. Good experimental results proving the feasibility of the proposed methods are also included.

key words: *computer art, stained glass image, data hiding, region growing, covert communication, watermarking, image authentication*

1. Introduction

In the era of computer networks, more and more people exchange digital images over the Internet. Many methods have been proposed to make digital images more artistic before publishing them. New types of computed-generated art image are also proposed. However, when such art images are transmitted over networks, people can duplicate or tamper with them easily. Thus, the issue of information protection must be considered seriously.

Many studies on automatic art image creation [1]–[12] and data hiding for information protection have been conducted. But rare of them integrate respective solutions to the two issues into a single approach. If one wants to create an art image to decorate his/her websites, e.g., and does not want the image to be downloaded and modified, additional *watermarking* works need be carried out to protect the copyright of the image. Also, when one wants to conduct *covert communication* via a secret message, one way is to hide the message into an art image which, by its artistic appearance, may detract a hacker from tampering with it. Or it might also happen that an art image published on the Internet is modified and claimed illicitly to be original. In such a case, *image authentication* is needed. In this study, an automatic method is proposed to create a new type of art image, which imitates the stained glass image often seen on church windows. Furthermore, a new method for *data hiding* via such art images is proposed, which may be adapted to solve all the three issues of watermarking for copyright protection [13], [14], covert communication [11], [14], and image authentication [15], [16] just mentioned.

About art image generation, Hertzmann [1] surveyed *stroke-based rendering* algorithms for creating non-photorealistic images which are composed by discrete elements like paint strokes and stipples. The main goal of these algorithms is to make generated images look like real artworks such as oil painting. Figure 1 shows an example of images created by Hertzmann [2], [3] where Fig. 1 (a) is a source painting and Fig. 2 (b) is the corresponding computer-generated oil painting image. Figures 2 (a) through 2 (d) shows some mosaic or tile images generated automatically by Hausner [4], Haeberli [5] and Matsumura et al. [6]. Recently, with the fast development of neural networks and deep learning applications, Chen, et al. [7] and Gatys, et al. [8] proposed image style transfer methods based on the use of convolutional neural networks to generate art images. The methods can transfer given images to specific styles like those created by famous artists, as shown in Figs. 2 (e) through 2 (j).

Stained glass windows are composed of glass pieces of different shapes, colors, and sizes. They first appeared in the 7th century, had heydays in the 16th century, and are still being built today [9]. Figure 3 (a) shows the detail of a

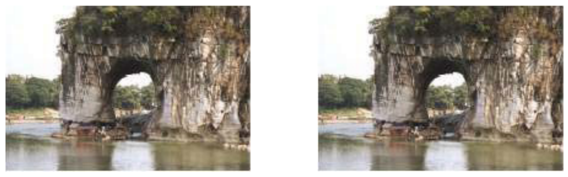


Fig. 1 Computer-generated oil-painting images. (a) The source painting. (b) An image imitating (a) created by Hertzmann [2], [3].

Manuscript received September 9, 2019.
Manuscript revised November 29, 2019.
Manuscript publicized January 15, 2020.

[†]The author was with the Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan.
[††]The author is with the Department of Computer and Communication Engineering, National Kaohsiung University of Science and Technology, Kaohsiung, Taiwan.
[†††]The author is with the Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan.
[††††]The author is also with the Department of Information Communication, Asia University, Taichung, Taiwan.
[*]Presently, with Somuch Co. Ltd., Taipei, Taiwan.
a) E-mail: dcwu@nkust.edu.tw (Corresponding author)
DOI: 10.1587/transinf.2019EDP7239

stained glass window in a church. Mould [10] proposed an algorithm for automatic creation of stained glass images like those in Figs. 3 (b) and 3 (c).

Briefly speaking, in the proposed method for stained glass image generation, a source image is processed by quantization and region growing to obtain a group of *glass regions* which looks like a stained glass version of the source image. In the proposed data hiding method, a technique of *slight glass cracking* is carried out to achieve the purpose of bit-code embedding in the image. The proposed data hiding technique can be modified further to fit the aforementioned three applications of information protection. It is mentioned by the way that the proposed method, differently from those that hide data in *natural images*, is proposed for applications on *art images* (here, on stained glass images) with the *specific* characteristics of the art image considered in detail in designing the data embedding and extraction algorithms.

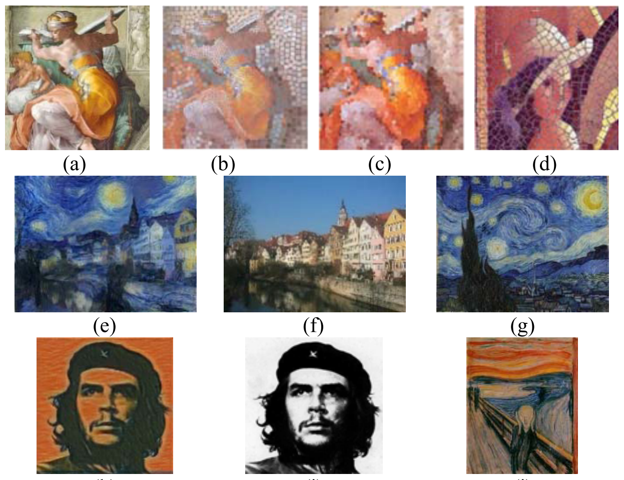


Fig. 2 Art images generated by computers. (a) A painting in Sistine Chapel by Michelangelo. (b) A mosaic image imitating (a) by Hausner [4]. (c) Another mosaic image imitating (a) by Haeberli [5]. (d) A tile image generated by Matsumura et al. [6]. (e) An art image generated by Chen, et al. [7] with (f) as the input according to V. van Gogh's post-impressionism art style of the painting "Starry Night" shown in (g). (h) An art image generated by Gatys, et al. [8] with (i) as the input according to E. Munch's expressionism art style of the painting "The Scream" shown in (j).

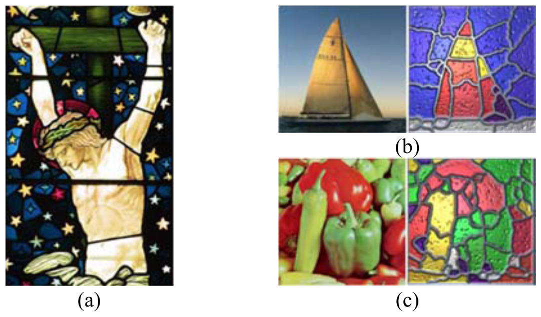


Fig. 3 Stained glass windows and computer-generated images. (a) A stained glass window of The Crucifixion, St. James Church, Staveley, UK. (b), (c) Two stained glass images generated by Mould [10].

This contrasts with those data hiding methods which usually just consider the *general* properties of natural images while conducting data embedding and extraction.

In the remainder of this paper, the proposed method for creating stained glass images is presented in Sect. 2. The proposed processes for data embedding and extraction via stained glass images, which implementing the proposed method are described in Sects. 3 and 4, respectively. The three applications of the proposed method mentioned previously are described in Sect. 5, with conclusions given in Sect. 6.

2. Automatic Creation of Stained Glass Images

A stained glass window is composed of multiple glass regions separated by black and thin gaps, called *leading*. A new type of art image, imitating the picture seen on the stained glass window, is proposed in this study from the viewpoint of data hiding in this study. It is called as usual *stained glass image* in this paper. In Sect. 2.1, a review of an existing method for stained glass image creation proposed by Mould [10] will be given. In Sect. 2.2, the ideas behind the method proposed in this study for automatic creation of the proposed new type of stained glass image are described; and an algorithm for implementing the ideas is presented in Sect. 2.3. Finally, some experimental results yielded by the proposed method are presented in Sect. 2.4.

2.1 Review of a Method for Stained Glass Image Creation

In Mould's method [10], at first an input image is segmented into regions which then are smoothed by image processing techniques. Next, the resulting regions are filled with colors chosen from a palette of heraldic tinctures, which are close to those of real stained glasses. Finally, a displacement-mapped plane representing imperfections in the regions is rendered, and leading is applied to the gaps between the region boundaries to obtain the final result.

Some results created by Mould [10] is shown in Fig. 4, where Fig. 4 (a) is the source image and Fig. 4 (b) is the seg-

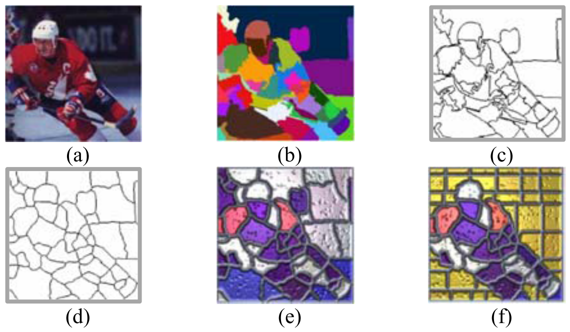


Fig. 4 An example of stained glass images created by Mould [10]. (a) An original image entitled Gretzky. (b) Segmented regions of (a). (c) Region boundaries of (b). (d) Smoothed region boundaries of (b). (e) A final version of generated stained glass images. (f) Another version of generated stained glass images with a different background.

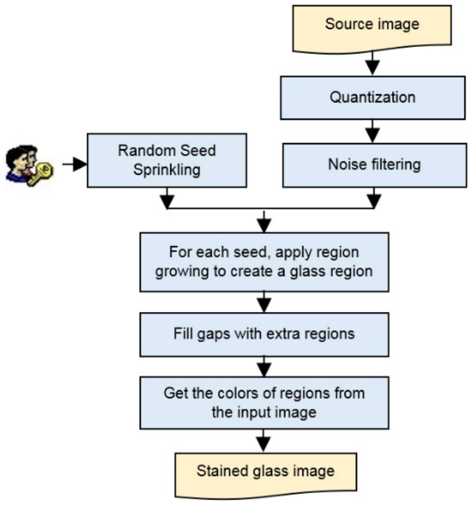

Source image
Quantization
Noise filtering
Random Seed Sprinkling
For each seed, apply region growing to create a glass region
Fill gaps with extra regions
Get the colors of regions from the input image
Stained glass image

Fig. 5 Proposed creation process of stained glass images.

mentation result shown as color regions. Figure 4 (c) is the region boundaries of Fig. 4 (b), and Fig. 4 (d) is a smoothed version of Fig. 4 (c). Figure 4 (e) is a final stained glass image, and Fig. 4 (f) is another whose background pattern and colors are different from those of Fig. 4 (e).

The appearances of the stained glass images created by Mould's method [10] are close to those seen on the real stained glass windows, but the method is not designed from the viewpoint of data hiding while the method proposed in this study instead is, though the generated stained glass images are less similar to the real ones. As examples for comparisons, some results yielded by the two methods with identical source images can be seen later in Fig. 11. Granted that the images generated by the proposed method are not so realistic as commonly-seen stained glass images, hopefully it may still be regarded as a new type of computer art via which data hiding can be conducted and detraction of the hacker' attention on the hidden data can be achieved!

2.2 Ideas of the Proposed Method

The proposed method for creating a stained glass image from a given source image is illustrated by the flowchart shown in Fig. 5. At first, the source image is quantized to yield simple color regions. Then, noise in the resulting image is eliminated by a voting filter. Subsequently, discrete pixels called *seeds* are sprinkled over the color regions randomly. Randomization of the seeds is adopted mainly for keeping the security of the embedded data in the stained glass image. Then, region growing is applied to each seed to obtain a group of *glass regions*. In this process, a *glass region tree* is constructed. Finally, large gaps between the glass regions are filled with extra glass regions by a *gap filling process*. As a result, a stained glass image is created. More details of these steps are described next.

(A) Preprocessing of the source image
The preprocessing operations applied to the input image are

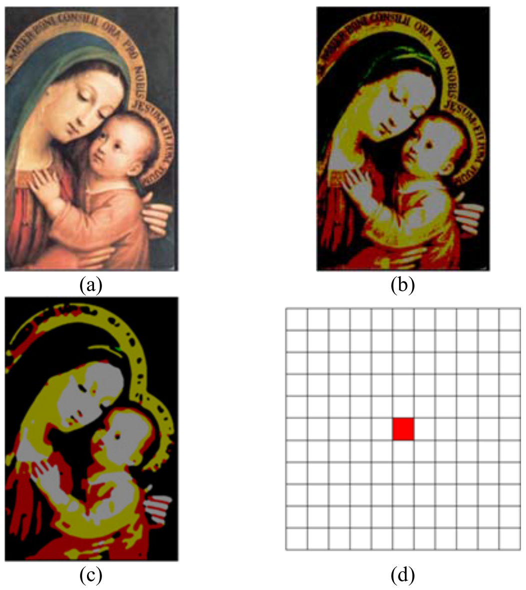


Fig. 6 Image preprocessing before region growing. (a) A source image, (b) a quantized image of (a). (c) A filtered image of (b). (d) A square voting filter of size 11×11.

quantization and filtering. Take Fig. 6 as an example, where the R, G, and B values of each pixel's color in Fig. 6 (a) are quantized into three bits, with one bit for each color channel. The result is shown in Fig. 6 (b), in which the color regions are seen to be shattered. This will affect the result of region growing. So, a 11×11 *voting filter* as shown in Fig. 6 (d) is applied to each pixel in Fig. 6 (b) which accumulates the numbers of pixels for each color within the filter scope, and resets the pixel's color to be that with the maximum accumulation value. Figure 6 (c) is the filtering result with the noise removed. Now, region growing can be applied to Fig. 6 (c) to obtain a group of color regions, called *glass regions*.

(B) Tree structure for region growing
In more detail, each glass region formed by region growing is obtained through the construction of a *glass region tree* of a structure as illustrated by Fig. 7 (a). Specifically, a node in the tree is either an *internal node* or a *leaf node*. An internal node has at least one *child node* and a leaf node does not. In addition, a tree node may also be classified as an *expanding node* or a *succeeding node*. An expanding node is the *first* node or the *last* one at a tree level (i.e., the *rightmost* or *leftmost* node when the tree level is laid out *horizontally* as shown in Fig. 7 (a)), and a succeeding node is one in between. For example, in Fig. 7, each expanding node is enclosed by red thick border lines and each succeeding node by black thin border lines. Also, as illustrated by Fig. 7 (a), each expanding node has, if any, two child nodes, one being an expanding node and the other a succeeding one. On the other hand, a succeeding node has, if any, only a child node which is a succeeding node as well.

(C) Construction process of glass region trees
While constructing a glass region tree T, the nodes grown

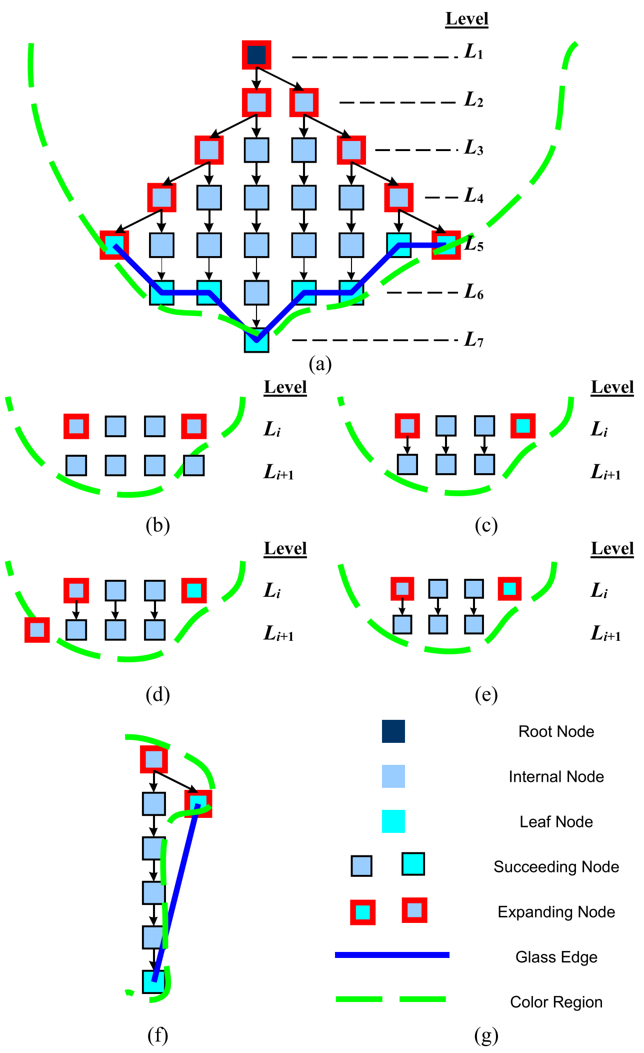


Fig. 7 Structure and expansion of the glass region tree. (a) A tree of a glass region; (b)–(f) Examples for illustrating details of Steps 2 and 3 in Algorithm 1. (g) Figure descriptions. Note (explanations of the tree levels L_1–L_7) – L_1: the root of glass region tree; L_2: the start level of region growing with two expanding nodes (in each of the four directions); L_3–L_7: intermediate levels of tree growing (with the number of levels depending on the size of the color region being grown).

of T correspond to the pixels in a preprocessed image S. So, e.g., "adding two expanding nodes N_1 and N_2 to a node N_0 as its two child nodes" means finding two pixels P_1 and P_2 in S corresponding respectively to N_1 and N_2 with each of P_1 and P_2 being a neighbor of the pixel P_0 corresponding to N_0 in S and being corresponding to an expanding node for further tree growing. Thus, in the sequel, we will discuss tree nodes directly, regarding "tree node" and "image pixel" as two equivalent terms, so long as the pixel has the property required by the tree node. The details of glass region tree construction are described as an algorithm as follows.

Algorithm 1: Constructing a glass region tree.

Input: a color region G in a preprocessed image S and a pixel P_0 in G.

Output: a glass region tree T for color region G with pixel P_0 as its root R.

Steps:

1. Take pixel P_0 as the root R of a tree T; and at the highest tree level L_1 of T, add two expanding nodes to tree level L_2 as the child nodes of R, and assign them as internal.
2. At each tree level L_i for $i \geq 2$, perform the following steps.
   - 2.1 For each internal node N_j, generate *horizontally or vertically* a child node C_j in image S, assign it as both internal and succeeding, and perform the following two steps.
     - (i) If N_j and C_j are not both in region G, then regard C_j as out of bound, discard it, and re-assign N_j as a leaf node.
     - (ii) If the depth of C_j is greater than that of its neighboring sibling, then regard C_j as inappropriate, discard it, and re-assign N_j as a leaf node.
   - 2.2 For each node N_j' which is both internal and expanding, generate *diagonally* a child node C_j' in image S, assign it as internal and expanding, and perform the following two steps.
     - (i) If N_j' and C_j' are not both in region G, then regard C_j' as out of bound, and discard it.
     - (ii) If the depth of C_j' is greater than that of its neighboring sibling, then regard C_j' as inappropriate, and discard it.
   - 2.3 If no child node is generated in Steps 2.1 and 2.2, terminate the tree growing process and exit; else, increase index i by 1, and go back to Step 2.

The major steps of Algorithm 1 above for constructing a glass region tree are explained here using Fig. 7 as an illustrative example. In Fig. 7, a dashed green curve is used to enclose the pixels of a color region derived from the preprocessing process. Initially in Step 1, two expanding nodes are added at tree level L_1 as the child nodes of the tree root R, and regard them as internal nodes.

In Step 2.1 of Algorithm 1, for each internal nodes at tree level L_i ($i \geq 2$), a child node which is internal and succeeding is generated. Then, Step 2.1(i) is performed, yielding a result which can be illustrated by Figs. 7 (b) and 7 (c). As shown in Fig. 7 (b), four succeeding nodes are generated at level L_{i+1}, in which the rightmost one is located, differently from its parent, in a color region outside the dashed green curve. So, according to Step 2.1(i), it is regarded as out of bound and is discarded, with its parent node being transformed into a leaf node instead of being an internal one as shown by Fig. 7 (c).

In Step 2.2, for each internal node which is also an expanding one at tree level L_i, a child node which is both internal and expanding is generated. Then, Step 2.2(i) is performed, yielding a result which can be illustrated by Figs. 7 (d) and 7 (e). As shown in Fig. 7 (d), only one expanding node is generated at level L_{i+1} which is the leftmost node at level L_{i+1}; the rightmost node at L_i, though expanding in nature originally, has been transformed to be a leaf node in Step 2.1(i), and so cannot be extended to generate child nodes at level L_{i+1}. However, as shown in Fig. 7 (e), the generated leftmost node at level L_{i+1} is located in a color region differently from its parent node; therefore, it is discarded *without* transforming its parent node to be a leaf node according to Step 2.2(i).

Steps 2.1(i) and 2.2(i) as illustrated can be seen to have a function of keeping the depth differences of neighboring leaf nodes smaller than or equal to one. The reason for designing them in this way is illustrated by a counter example shown in Fig. 7 (f) where the depth difference of the two neighboring leaf nodes is four, and the blue line is the *edge* defined by the two leaf nodes. It can be seen that this edge *overlaps* another color region outside the dashed green curve. This will result in yielding overlapping glass regions after region growing. That is why we have to keep the depth differences of neighboring leaf nodes to be smaller than or equal to one by carrying out Steps 2.1(ii) and 2.2(ii).

Finally, in Step 2.3, we decide to terminate the tree growing process if no child node is generated in Steps 2.1 and 2.3. After the process terminates, the pixel group corresponding to the derived tree nodes will become part of a glass region yielded by the proposed glass region growing process described next.

2.3 Proposed Glass Region Growing Process

After preprocessing the source image, a color region image like Fig. 6 (c) is derived. Then, region growing is started. The ideas behind this process, an algorithm to implement them, and a gap filling scheme are described in the following.

(A) Ideas of the proposed glass region growing process

After seed sprinkling, at first four trees are ‘rooted’ at each seed pixel, which are denoted by *Nt*, *Et*, *St*, and *Wt* for the four directions of north, east, south, and west, respectively, as shown in Fig. 8. This work is accomplished by forming an *initial glass range* of 3×3 nodes with the seed pixel as the center, which is shown in Fig. 8 (a) or 8 (c) as the central square enclosed by the pink boundary lines. And starting from the top-middle node and around the eight boundary nodes of the initial glass range, every two boundary nodes are taken sequentially to form an initial node pair of each of the four trees of *Nt*, *Et*, *St*, and *Wt* in order, which are shown as the purple, pink, blue, and orange node pairs in the initial glass range in Fig. 8 (a) or 8 (b). Then, the four trees are grown in the four directions, respectively, by applying Algorithm 1. The trees of all seeds are grown simultaneously level by level. The resulting four trees are shown by different colors in Fig. 8.

Subsequently, the leaf nodes in each tree and then the four trees are *linked* integrally to form the boundary of a complete glass region. The links of the leaf nodes are drawn as blue lines and those of the trees as yellow lines, as shown in Fig. 8. Specifically, a *leaf node link* is created simply by linking every pair of adjacent leaf nodes in each tree. On

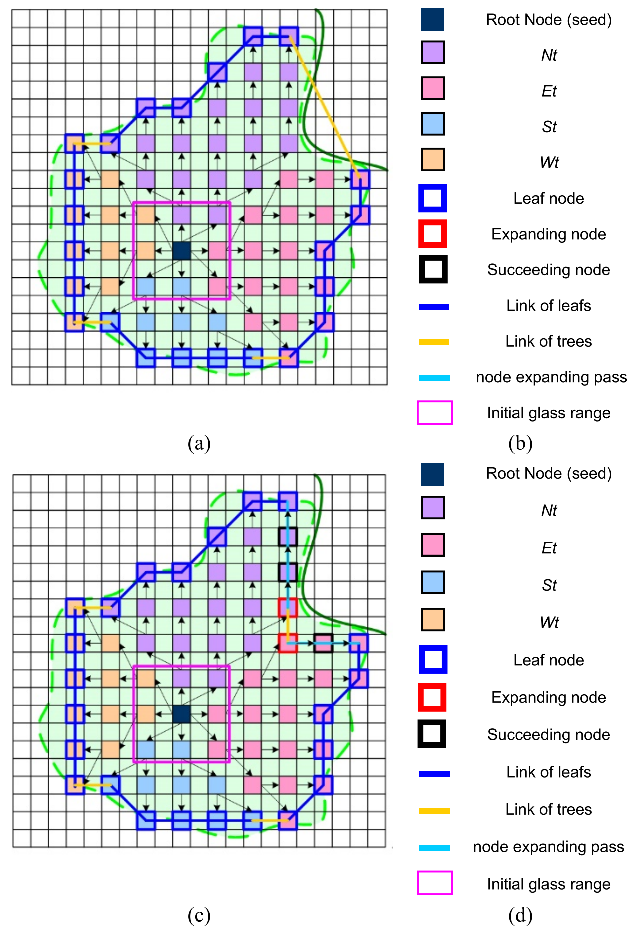


Fig. 8 Illustration of region growing. (a) Linking leaf nodes. (b) Notations. (c) Linking trees. (d) Notations identical to (b). Note (explanations of the notations) – (1) root node: a node on which a seed is sprinkled and from which region growing starts; (2) leaf node: a node having no child node and lying on the boundary of the region; (3) internal node: a node which is not a leaf node; (4) expanding node: the first or last node at a tree level (i.e., the rightmost or leftmost node when the tree level is laid out horizontally); (5) succeeding node: a node between two expanding nodes. (6) *Nt*, *Et*, *St*, and *Wt*: the north, east, south, and west trees starting from the root node.

the contrary, the four trees are *not* linked simply by linking the *rightmost* and *leftmost* leaf nodes of two adjacent trees because a link so created may be out of the color region, as illustrated by Fig. 8 (a) where the link between the two neighboring trees *Nt* and *Et* is out of the color region, leading to glass-region overlapping in the final region growing result. Note that in the above discussions and those in the sequel, it is assumed that each tree has *its root located on the top*, so that the terms rightmost and leftmost can be used; if not so, it is assumed that the tree has been rotated to be so.

In order to solve the aforementioned glass-region overlapping problem, at first the *node expanding pass (NEP) of a given leaf node* is defined, which is an *upward* trace of all the expanded nodes from the leaf node to the tree root. These nodes in the NEP are all grown in the tree growing process described by Algorithm 1. Then, the problem is solved by a scheme consisting of the following three steps.

(i) Find in an upward direction the *first expanding node* N_{1e} in the NEP P_1 of the *leftmost* leaf node N_{1f} of a tree T_1 (like *Nt* in Fig. 8 (c)), and construct a *partial NEP* P_1' of P_1 consisting of all the nodes traced from the expanding node N_{1e} to the leaf node N_{1f}.

(ii) Do similarly to find in an upward direction the *first expanding node* N_{2e} in the NEP P_2 of the *rightmost* leaf node N_{2f} of the *clockwisely-neighboring* tree T_2 (like *Et* in Fig. 8 (c)), and construct a *partial NEP* P_2' of P_2 consisting of all the traced nodes from the expanding node N_{2e} to the leaf node N_{2f}.

(iii) Regarding all the nodes in each of the partial NEP's P_1' and P_2' as already being linked together, link the two trees T_1 and T_2 by linking N_{1f} and N_{2f} which connect to the two partial NEP's P_1' and P_2', respectively, resulting in a *chain link* of $P_1' - N_{1f} - N_{2f} - P_2'$ for use as part of the final glass region boundary (like the L-shaped boundary appearing at the concave corner at the middle right side of the glass region in Fig. 8 (c)).

The above steps together will be called in the sequel the *tree linking scheme* for a pair of clockwisely-neighboring trees (i.e., T_1 and T_2 in the above description).

Two points need be noted in the above region growing process. First, according to the proposed glass region tree growing process described by Algorithm 1, each glass region has a minimum size of 3×3, or equivalently, nine nodes, as shown by the pink squares enclosing the initial glass range in the middle of Fig. 8 (a) or 8 (c). Therefore, in order to prevent the resulting glass regions from overlapping one another, while conducting seed sprinkling, we have to keep the inter-seed distance at least *three nodes* in magnitude. Furthermore, in this study we let the distance between nodes to be *two pixels*. Therefore, the minimum inter-seed distance must be *six pixels*.

The second point to be noticed is that more than one seed might be located undesirably in a color region after seed sprinkling. In order to prevent this from happening, a global *tree map* is created for use in the proposed glass region growing process to record the positon of each generated seed in order. And after the first seed is recorded in the map, every seed generated later at a certain position is checked against the map to see if a formerly-generated seed has been sprinkled into an identical color region already; if so, then discard the seed and generate another until the collision case is avoided. Furthermore, the position of each node generated in the glass region tree construction process is also recorded in the map. And if a parent node, which is internal originally, 'wants' to extend a child node into an *occupied* node position in the map, then the status of the parent node is transformed from an internal node to a leaf node. Finally, it is mentioned that the tree map will also be used for the gap filling process described next.

(B) Proposed glass region gap filling process

Though the problem of multiple seeds sprinkled in an identical color region can be solved as described above, it might happen contrarily that no seed is sprinkled in certain regions, causing *gaps* between the grown regions. For example, the blue polygons in Fig. 9 are the regions yielded by the glass region growing process, and it can be seen that between them there exist gaps which should be filled up further.

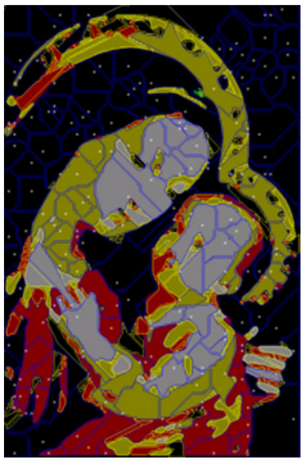


Fig. 9 Illustration of the gap filling process.

In this study, for gap filling, initially the tree map is scanned. If no tree node is found within a certain square area of 15×15 pixels, an additional seed is put at the center of that square and the previously-proposed glass region growing process is applied on it. For example, the yellow polygons in Fig. 9 are the additional glass regions which are found by such a gap filling process.

(C) Proposed glass region growing process

The proposed glass region growing process realizing all the above-mentioned ideas is described as an algorithm below.

Algorithm 2: Glass region growing to create a stained glass image.

Input: a source image S and a secret key K.

Output: a stained glass image G.

Steps:

Stage 1 — initialization (including preprocessing, tree map setup, and random seed sprinkling).

```
1. Preprocess the source image $S$ by quantization and noise filtering to obtain a simplified image $S'$.
2. Set up a tree map $M$ of the same size as that of $S'$ with cells to record generated seeds and nodes, initially empty.
3. Use a random number generator $R$ with the secret key $K$ as the input to sprinkle randomly seeds with the minimum inter-distance of 6 pixels into $S'$ in the following way.
   3.1 For each generated seed $d$ at random position $(x, y)$, check the content of the tree map $M$ at $(x, y)$, and if there already exists a seed there, then discard $d$; else, save seed $d$ into $M$ at $(x, y)$.
   3.2 Repeat Step 3.1 until the number of saved seeds equal to a pre-selected threshold $T_d$.
```

Stage 2 — constructing glass region trees.

```
4. For each sprinkled seed $d$, perform the following steps
```

to yield a glass region R_d.

4.1 Identify the color region R where seed d is located.

4.2 Regard d as the root of four glass region trees labeled as *Nt*, *Et*, *St*, and *Wt* in the clockwise direction and construct an initial part of each tree in the following way.

(i) Form an *initial glass range* R_{init} of 3×3 nodes with d as the center.

(ii) Starting from the top-middle node and around the eight boundary nodes of R_{init}, take sequentially every two boundary nodes to form an initial node pair of each of *Nt*, *Et*, *St*, and *Wt* in order.

4.3 Perform Algorithm 1 with R and d as the input to grow each of the four trees.

4.4 Link the four trees, *Nt*, *Et*, *St*, and *Wt*, in the following way to produce the desired glass region R_d.

(i) Link every pair of adjacent leaf nodes of each of the four trees.

(ii) For every pair (T_1, T_2) of clockwisely-neighboring trees of the four trees (i.e., for every pair of (Nt, Et), (Et, St), (St, Wt), and (Wt, Nt)), perform a tree linking scheme in the following way to link T_1 and T_2.

(1) Trace upward all the expanded nodes from the leftmost leaf node $N_{1\text{f}}$ of T_1 to the root node d to construct the node expanding pass (NEP), P_1, *of* $N_{1\text{f}}$; and find the first expanding node $N_{1\text{e}}$ in P_1 to trace all the nodes from $N_{1\text{e}}$ to $N_{1\text{f}}$ to construct a *partial NEP* P_1' of P_1.

(2) Do similarly to (1) to construct the NEP, P_2, of the rightmost leaf node $N_{2\text{f}}$ of T_2 and find the first expanding node $N_{2\text{e}}$ in P_2 to construct a partial NEP P_2' which includes all the traced nodes from $N_{2\text{e}}$ to $N_{2\text{f}}$.

(3) Regarding all nodes in each of the partial NEPs P_1' and P_2' as already being linked together, link T_1 and T_2 by linking $N_{1\text{f}}$ and $N_{2\text{f}}$ which connect to P_1' and P_2', respectively, resulting in a *chain link* of $P_1' - N_{1\text{f}} - N_{2\text{f}} - P_2'$ which can be used as part of the final glass region boundary.

(iii) Take all the nodes of the four linked trees as the desired glass region R_d.

Stage 3 — conducting gap filling and composing the desired stained glass image.

5. Scan the tree map to find any square area of size 15 × 15 with no tree node filled, and conduct the following steps for each of the square areas so found.

5.1 Put a seed into the center of the square area.

5.2 Perform Steps 4.1 through 4.4 to yield a glass region.

6. Compose the desired stained glass image G in the following way and exit.

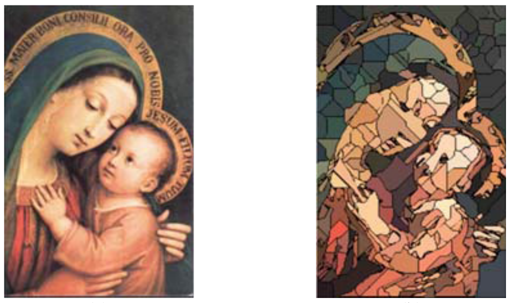


Fig. 10 An experimental result. (a) A source image. (b) A stained glass image created with (a) as the input.

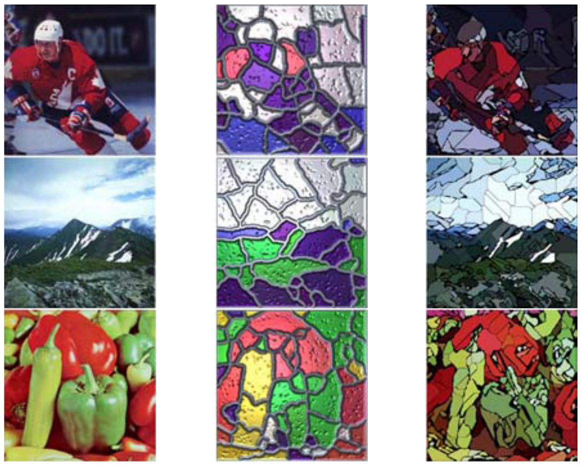


Fig. 11 Some experimental results with the left column including the source images, the middle including the images created by Mould [10], and the right including those created by the proposed method.

6.1 Copy into G those pixels in source image S covered by all the glass regions yielded in Steps 4 and 5.

6.2 Apply leading to those pixels in G (i.e., color them by black) which are not filled up in Step 6.1.

In the next section, Sect. 3, the proposed data hiding techniques for uses in the three applications of watermarking, secret communication, and image authentication, will be presented. The data are embedded into the glass region generated with randomly scattered seeds, but not into the glass regions which are generated by gap filling except in the application of image authentication.

2.4 Experimental Results and Discussions

Figure 10 is a result of applying Algorithm 2 above, where Fig. 10 (a) is the source image and Fig. 10 (b) the yielded stained glass image in which it can be seen that the glass regions do not overlap one another. This characteristic is good for data embedding and extraction later. More experimental results are shown in Fig. 11 where the images in the first column are the source, those in the middle are the stained glass images created by Mould's method [10], and those in the last created by the proposed method.

3. Data Hiding in Stained Glass Images

In this section, the proposed data hiding method in a glass region is presented, whose basic ideas are described in Sect. 3.1; and the hiding technique, which is based on a scheme of tree node number modification, is proposed in Sect. 3.2. An involved issue is the acquisition of *effective* trees, which is described in Sect. 3.3 with a solution proposed. Finally, an algorithm to implement these ideas is presented in Sect. 3.4.

3.1 Basic Ideas of Proposed Data Hiding Method

The feature utilized in this study for data hiding via stained glass images is *the number of tree nodes* in a glass region tree; by removing a computed number of the *deepest* tree nodes, message data can be hidden in a stained glass image. Though this will cause little cracks at the edges and corners of the glass regions, the change in the resulting image appearance is still acceptable visually. For example, as illustrated by Fig. 12 (a), the two nodes on the top and enclosed by red boundaries are the deepest nodes of the north tree *Nt* of a glass region, which can be removed to embed data in the proposed data hiding method. Figure 12 (b) shows the result of removing them, creating a little crack in the glass region represented by the tree nodes.

Although four trees are enclosed in a glass region, *not* all of them can be used for data hiding; the number of nodes in a tree must be *large enough* so that the data can be embedded by removing some tree nodes without causing relatively intolerable crack distortions. We name the trees that can be used for data hiding *effective trees*. More details on effective trees will be discussed subsequently, and so will be the glass region feature detection process.

The overall concept of embedding data into a stained glass image is illustrated by the flowchart shown in Fig. 13. Since only one type of glass feature is used for data hiding, only one of the three aforementioned applications of secret communication, watermarking, and image authentication can be implemented at a time. No matter what kind of data to be embedded is, the input data are first transformed into a bit-code sequence in advance. Then, the previously-described stained glass image creation process is performed with glass region trees yielded. Afterwards, the bit sequence of the data is embedded by removing some deepest nodes of every single effective tree. More details are discussed next.

3.2 Data Embedding by Tree Node Number Modification

The core concept of data hiding by tree node number modification proposed in this study can be illustrated by Fig. 14. Assume that a message is to be embedded into a glass region tree T with TNN nodes. Initially, the remainder, REM, of dividing the number, TNN, by a divisor, DIV, is computed,

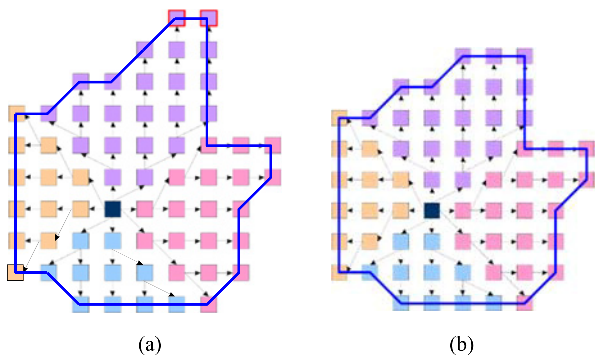


Fig. 12 An example of removing nodes for data hiding. (a) A glass region copied from Fig. 8 (b) with two nodes (enclosed by red boundaries on the top) to be removed. (b) A glass region with data to be embedded by removing two nodes.

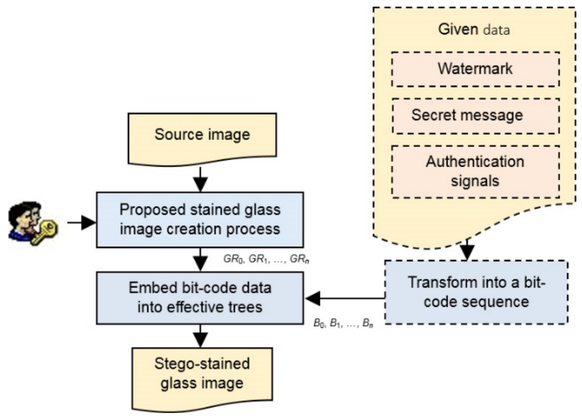

Given data
Watermark
Secret message
Authentication signals
Source image
Proposed stained glass image creation process
$GR_0, GR_1, \ldots, GR_n$
Embed bit-code data into effective trees
Transform into a bit-code sequence
$B_0, B_1, \ldots, B_n$
Stego-stained glass image

Fig. 13 Proposed process for embedding bit-code data into a stained glass image obtained from a source image.

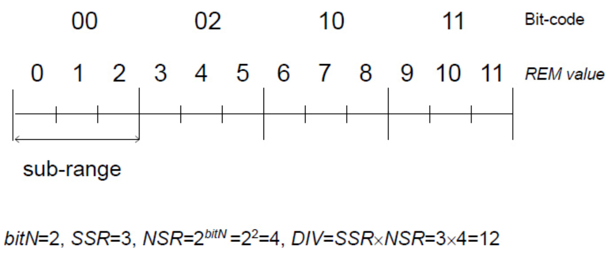

00
02
10
11
Bit-code
0 1 2 3 4 5 6 7 8 9 10 11
REM value
sub-range
$bitN$=2, SSR=3, $NSR=2^{bitN}=2^2=4$, $DIV=SSR\times NSR=3\times4=12$

Fig. 14 Core concept of data hiding via stained glass images.

where DIV is to be determined. The value of REM will thus range between 0 and $DIV - 1$ as shown by the following formula:

$$REM = TNN \bmod DIV,\ 0 \le REM < DIV - 1. \tag{1}$$

Then, the value range of REM is divided into several sub-ranges, with each sub-range representing a specific bit-code. In the case of Fig. 14, the bit-code is assumed to contain two bits, so the range is divided into four sub-ranges representing the bit-codes 00, 01, 10 and 11. In other words, the number of sub-ranges, NSR, is computed by the following formula:

$$NSR = 2^{bitN}, \tag{2}$$

where *bitN* is the number of bits to be embedded into tree T and is a pre-selected parameter. It is noted here that because each sub-range includes several values of *REM*, a certain degree of tolerance of detection errors can be achieved. Take Fig. 14 as an example, where there are four sub-ranges, and each sub-range includes three values of *REM*. If an *REM* falls, for example, into the first sub-range which includes the three values 0, 1, and 2, then no matter what value *REM* is, as long as it is one of the three values, it means that the bit-code 00 is embedded into tree T. Now, the value of *DIV* can be determined as follows:

$$DIV = SSR \times NSR, \tag{3}$$

where *SSR* is the number of *REM* values spanned by a sub-range and *NSR* is the number of sub-ranges decided by Eq. (2) above. So, the value of *DIV* in Fig. 14 is $3 \times 4 = 12$.

Now, to embed a bit-code C into tree T, in this study it is proposed that the value *REM* be adjusted to be the central value of the target sub-range which represents the bit-code C. This way is considered also helpful to allow the computed *REM* value to have an even larger tolerance of errors to represent a bit-code (i.e., the computed *REM* may be any of those values slightly larger or smaller than the central value in the target sub-range without causing errors of bit-code representations). This means that the number *NNR* of nodes to be removed from tree T should be computed by

$$NNR = |REM - \textit{central value of target sub-range}|. \tag{4}$$

For the example of Fig. 14, assume that the value *REM* is 9 and the bit-code to be embedded is 01. The target sub-range representing bit-code 01 is the second one whose central value is 4. So the number of nodes to be removed from tree T, *NNR*, for embedding bit-code 01 is computed as $NNR = |9 - 4| = 5$. By removing the five deepest nodes from tree T, the new *REM* will become the central value, four, of the second sub-range, which still represents the bit-code 01.

3.3 Acquiring Effective Trees for Data Hiding

Before data embedding via a stained glass image can be conducted, for each tree T of a glass region, it needs to know if data can be embedded into T, or equivalently, to know if the tree is *effective* with a sufficient number of nodes to be removed for the data hiding purpose. Specifically, as discussed previously, with the parameter *DIV* fixed, at most $DIV - 1$ nodes are removed from a target tree. Also, each tree in a glass region contains at least three nodes in the proposed creation process as mentioned previously. Therefore, an effective tree must contain at least $DIV + 3$ nodes for a bit-code to be embedded in all cases. We call this minimum number of nodes in an effective tree for data embedding, the *minimum tree size*, and denote it by *minTSE*. Also, to prevent glass regions from being broken into tiny pieces, we let the value of *minTSE* be $DIV \times 2 + 3$ instead of just the above-mentioned value of $DIV + 3$.

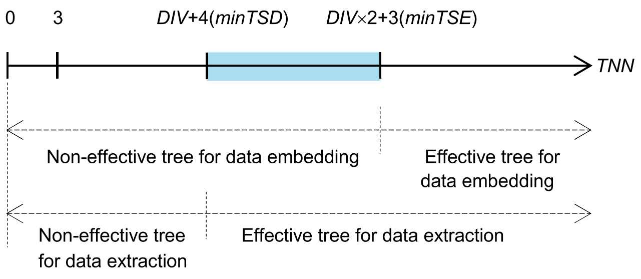

0
3
DIV+4(minTSD)
DIV×2+3(minTSE)
TNN
Non-effective tree for data embedding
Effective tree for data embedding
Non-effective tree for data extraction
Effective tree for data extraction

Fig. 15 Concept of acquiring effective trees in data embedding and extraction.

On the other hand, during data extraction, because the value of *minTSE* now is taken to be $DIV \times 2 + 3$ and because at most $DIV - 1$ nodes are removed from a target tree for data embedding, the minimum number of nodes in an effective tree for data extraction, which is denoted by *minTSD* subsequently, should thus be

$$(DIV \times 2 + 3) - (DIV - 1) = DIV + 4.$$

In other words, if the number *TNN* of nodes of a tree is *not smaller* than the value *minTSD* above, it can be decided that there are data embedded in the tree.

Now, it might happen that a tree has its number of nodes larger than *minTSD* and smaller than *minTSE*, as shown in the blue area of Fig. 15, then it might occur that no message is embedded in the data embedding process but some meaningless bit-codes are extracted out in the data extraction process. To avoid such a case, it is proposed to reduce the value of *TNN* to be *minTSD* by removing some of the deepest nodes of the target tree during the process of acquiring effective trees. Then, there will be no tree with its value of *TNN* between *minTSD* and *minTSE* (i.e., in the blue area in Fig. 15) after the effective tree acquisition process.

3.4 Data Embedding Algorithm

The above discussions about the proposed data embedding process are summarized as an algorithm in the following.

Algorithm 3: Embedding bit-code data into the glass regions of a stained glass image.

Input: a sequence *GR* of n glass regions of a source image S; a sequence B of m bit-codes of a message M with each bit-code having *bitN* bits; and two pre-selected parameters *minTSE* and *minTSD* representing the minimum tree size and the minimum number of nodes in an effective tree, respectively.

Output: a stego-stained glass image G with B embedded.

Steps:

1. Append an ending bit-code pattern B_{m+1} to the end of the sequence B of bit-codes, resulting in a new sequence $B' = \{B_0, B_1, \ldots, B_m, B_{m+1}\}$ with each B_i having *bitN* bits.
2. Retrieve the four glass region trees from each GR_i in the input sequence of glass regions, $GR = \{GR_1, GR_2, \ldots, GR_n\}$; and concatenate the resulting $4n$ trees

```
   into a sequence, $RT = \{RT_0, RT_1, \ldots, RT_{4n}\}$.
3. For each $RT_i$ in $RT$ with $TNN_i$ nodes, perform one of
   the following two cases to acquire a sequence of effec-
   tive trees, $ET = \{ET_0, ET_1, \ldots, ET_k\}$, where $k \leq 4n$:
   3.1 if $TNN_i > minTSE$, then add $RT_i$ into $ET$ as $ET_i$;
   3.2 if $minTSD \leq TNN_i < minTSE$, then
       (i)   let $n = TNN_i - minTSD$;
       (ii)  remove the $n$ deepest nodes of $RT_i$ so that
             $TNN_i = minTSD$; and
       (iii) add the resulting $RT_i$ into $ET$ as $ET_i$.
4. For each pair of $B_i$ and $ET_i$, $i = 1, 2, \ldots, m+1$, perform
   the following steps.
   4.1 Obtain the value $REM_i$ for $ET_i$ according to
       Eqs. (1) through (3) by the following steps:
       (i)   count the number $TNN_i'$ of nodes in $ET_i$;
       (ii)  compute the number $NSR$ of sub-ranges of
             the range $RG_i$ of $REM_i$ as $NSR = 2^{bitN}$;
       (iii) decide the number $SSR$ of $REM_i$ spanned by
             a sub-range of range $RG_i$;
       (iv)  compute the divisor $DIV$ for $REM_i$ as

                 $DIV = SSR \times NSR$;

       (v)   compute the remainder $REM_i$ as

                 $REM_i = TNN_i' \bmod DIV$

             so that $0 \leq REM_i < DIV - 1$.
   4.2 Compute the corresponding number $NNR_i$ of
       nodes to remove from $ET_i$ according to Eq. (4) as

             $NNR_i = |REM_i - CV_i|$

       where $CV_i$ is the central value of the target sub-
       range determined by the bit-code $B_i$.
   4.3 Remove the deepest $NNR_i$ nodes from $ET_i$.
5. Construct the desired stego-stained glass image $G$, ini-
   tially empty, in the following way and exit.
   5.1 Copy into $G$ those pixels in the source image $S$
       which correspond to the nodes of all trees in the
       resulting sequence $ET$ of trees.
   5.2 Apply leading to those pixels in $G$ (i.e., color them
       by black) which are not processed in Step 5.1.
```

Note that in the above algorithm, an ending bit-code pattern is appended to the end of the bit-code sequence, which is embedded as well into the stego-image. It will be used as the signal to end the bit-code extraction operations during the data extraction process as discussed next.

4. Extraction of Data from Stego-Images

In this section, after a description of the ideas behind the proposed data extraction process in Sect. 4.1, an algorithm implementing the ideas is presented in Sect. 4.2. Some experimental results are shown in Sect. 4.3.

4.1 Ideas of Proposed Method

A flowchart of the proposed process for data extraction from stego-stained glass images is shown in Fig. 16. The secret key is applied in the glass feature detection process, which, as done in the image creation process, is used for seed generation using a random number generator. A seed sequence is derived, which is identical to the one derived in the image creation process. The seed sequence is then sprinkled on the stego-image instead of on the color region image, differently from what is done in the data hiding process. Subsequently, the processes of glass region tree construction and glass region growing, as presented in Sect. 3, are conducted to yield a sequence of glass regions $GR = \{GR_0, GR_1, \ldots, GR_n\}$. The two sequences of glass regions derived in the image creation process and in the region growing process here presumably are identical. Also, the effective glass region trees $ET_0, ET_1, \ldots, ET_k$ are picked out. By counting the number REM_i of nodes in each single effective tree ET_i, the embedded bit-code sequence can be extracted.

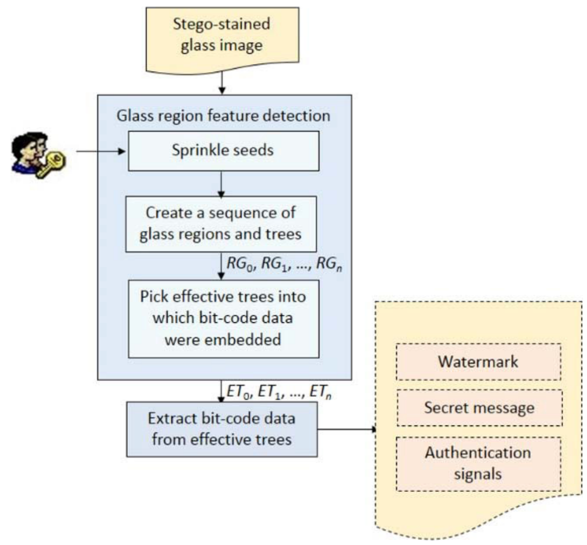

Stego-stained glass image
Glass region feature detection
Sprinkle seeds
Create a sequence of glass regions and trees
$RG_0, RG_1, \ldots, RG_n$
Pick effective trees into which bit-code data were embedded
$ET_0, ET_1, \ldots, ET_n$
Extract bit-code data from effective trees
Watermark
Secret message
Authentication signals

Fig. 16 Proposed process for extracting data from a stego-image.

4.2 Data Extraction Algorithm

An algorithm to implement the above ideas of data extraction is presented in the following.

Algorithm 4: Extracting hidden data from a stego-stained glass image.

Input: a stego-stained glass image G into which the sequence B of bit-codes of a message M was embedded with each bit-code having $bitN$ bits; and the secret key K used in the data embedding process.

Output: the sequence B of bit-codes of M.

Steps:

```
1. With the stego-image $G$ and the secret key $K$ as inputs,
   perform Algorithm 2 to generate a sequence of glass re-
   gions $GR = \{GR_0, GR_1, GR_2, \ldots, GR_n\}$ with each $GR_i$
   including four glass region trees.
2. Retrieve the four glass region trees from each $GR_i$, $i =$
   $1, 2, \ldots, n$, and concatenate the resulting $4n$ trees into
```

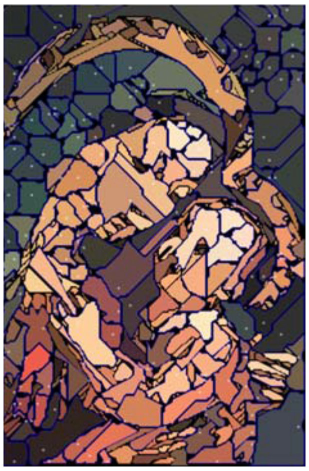


Fig. 17 Detection result of a stained glass image.

a sequence, $RT = \{RT_0, RT_1, \ldots, RT_{4n}\}$.

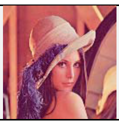
3. Perform Step 3 of Algorithm 3 to acquire a sequence of effective trees, $ET = \{ET_0, ET_1, \ldots, ET_k\}$, from RT where $k \le 4n$.
4. From each ET_i, $i = 1, 2, \ldots, k$, perform the following steps to construct a sequence B of bit-codes, initially empty.
   4.1 Count the number of nodes in ET_i and take it as the value of REM_i.
   4.2 Let the number of values spanned by a sub-range of the range RG_i of all possible values of REM_i be denoted by SSR which is pre-selected.
   4.3 (*Determine which sub-range the value REM_i falls in*) Compute the decimal value $C_i = \lceil REM_i/SSR \rceil$ where $\lceil\cdot\rceil$ is the integer ceiling function; and decide that REM_i falls in the C_i-th subrange of range RG_i.
   4.4 Transform C_i into a bit-code B_i of $bitN$ bits.
   4.5 If B_i is not the ending bit-code pattern which is assumed to be known in advance, then put B_i into B in order; else, exit with B as the desired sequence of bit-codes of the message M.

4.3 Experimental Results

Figure 17 shows the result of glass regions found by Algorithm 4 with a stego-image as the input. The white points sprinkled on the image are the seeds generated by the secret key. The blue polygons are the found glass regions according to the seeds. As can be seen, they fit well the original color regions in the stego-image. Subsequently, the embedded data are extracted from the effective trees, into which message data have been embedded.

About the data embedding capacities of the proposed method, three images named 'Lena', 'Pepper', and 'Madonna and Child' with different sizes of 512 × 512, 600 × 800, and 773 × 1140, respectively, were tested with their embedding capacities being counted while running Algorithm 3 with input messages composed of random bit-codes. The results for a series of cases of sprinkling different numbers of seeds are shown in Table 1 and Fig. 18.

Table 1 Statistics of data embedding capacities of three tested images resulting from different numbers of sprinkled seeds.

No.	A	B	C
Title	Lena	Pepper	Madonna & Child
Image			
Size	512×512	600×800	773×1140
Max #regions	1764	3300	6080
# Glass regions (seeds)	Embedding capacity (#bits)	Embedding capacity (#bits)	Embedding capacity (#bits)
10	15	12	19
100	218	246	275
200	326	412	545
300	**345**	543	741
400	328	594	906
500	268	**623**	1043
600	229	618	1166
700	171	589	1252
800	142	552	1290
900	103	502	**1300**
1000	72	428	1297
1100	47	372	1271
1200	37	324	1238
1300	22	285	1176
1400	18	235	1140
1500	12	203	1090
1600	8	175	1022

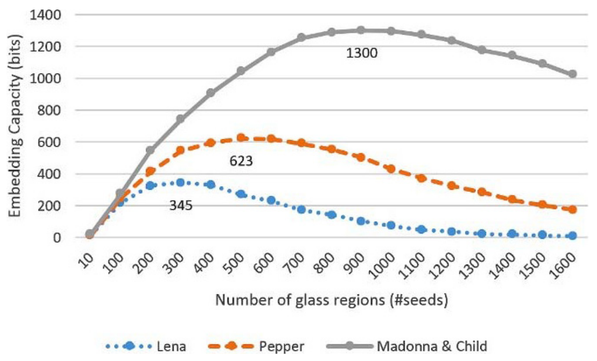


Fig. 18 Graphs showing trends of data embedding capacities of three tested images resulting from different numbers of sprinkled seeds.

It can be seen from the table and figure that the data embedding capacity is affected not only by the image size and content as is usually known but also *by the number N of sprinkled seeds* because each seeds grows to yield a glass region. Specifically, there exists an *optimal* number $N_{\max}$ of sprinkled seeds by which the resulting data embedding capacity becomes the *maximum*. For example, for the image of 'Lena' with size 512 × 512, the embedding capacity becomes the maximum value of 345 when $N_{\max} = 300$; for 'Pepper' with size 600 × 800, the maximum capacity of 623 occurs when $N_{\max} = 600$; and for 'Madonna and Child' with size 773×1140, the maximum capacity of 1300 occurs when $N_{\max} = 900$.

The reason why an optimal data embedding capacity will occur when a certain number $N_{\max}$ of seeds are sprinkled on each image is twofold: (1) if the number N of sprinkled seeds is too large, then the image will be divided into too many *small* glass regions so that regions that are large enough to be effective for secret bit embedding become very few, resulting in a small data embedding capacity; and (2) if the number N of sprinkled seeds is too small on the contrary, then the image will be divided into too few glass regions, though large enough, so that again there will yield too few glass regions to generate a sufficient number of effective regions, resulting again in a small data embedding capacity.

5. Data Hiding Applications

Three applications of the proposed data hiding technique via stained glass images are described in this section, namely, covert communication, watermarking for copyright protection, and image authentication.

5.1 Covert Communication via Stained Glass Images

Covert communication here means embedding a message data into a given stained glass image and transmitting the resulting stego-image to a receiver in a secret way. The aim is to protect the message from being intercepted by hackers.

(A) Secret Message Embedding

The proposed process for embedding a secret message into a stained glass image for covert communication is presented as an algorithm below.

Algorithm 5: Embedding a secret message in a stained glass image for covert communication.

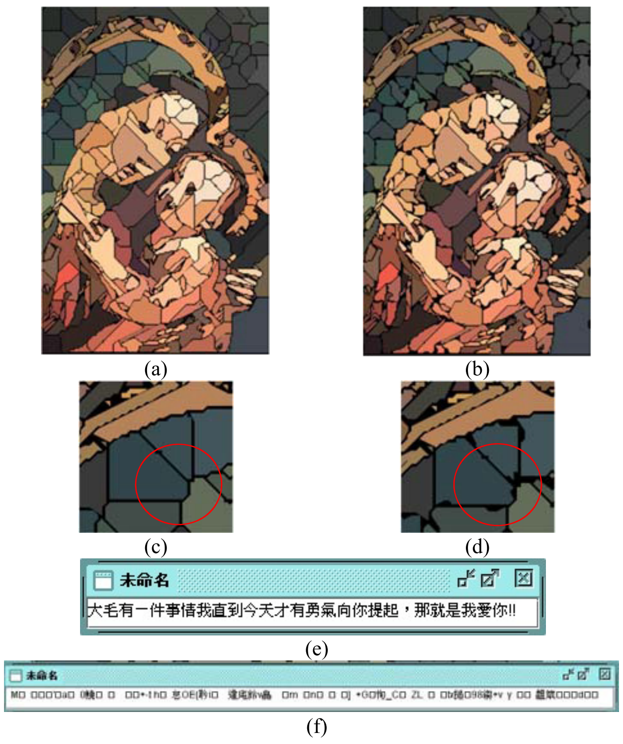
Input: a source image S, a secret message M, and a secret key K.

Output: a stego-stained glass image G with M embedded.

Steps:

1. Perform the steps in Stages 1 and 2 of Algorithm 2 with source image S and secret key K as the inputs to yield a sequence $GR = \{GR_0, GR_1, \ldots, GR_n\}$ of glass regions.
2. Transform the input message M into a sequence $B = \{B_0, B_1, \ldots, B_m\}$ of bit-codes, each with *bitN* bits.
3. Use the secret key K and a random number generator to randomize B, resulting in a randomized bit-code sequence $B' = \{B_0', B_1', \ldots, B_m'\}$.
4. Perform the data embedding algorithm, Algorithm 3, with sequences GR and B' as the inputs to yield a stego-stained glass image G as the desired output.

It is noted that Stage 3 of Algorithm 2, which mainly performs the work of gap filling, is not carried out in Step 1 of the above algorithm, Algorithm 5. This is owing to the reason that it is not desired to use those glass regions yielded by gap filling for data hiding, because the gaps change in size after data embedding, leading possibly to undesirable changes of the yielded glass regions during data extraction which in turn causes erroneous extracted bit-code data.



未命名
大毛有一件事情我直到今天才有勇氣向你提起，那就是我愛你!!

未命名
MD DDDDaD 0鰊D D DD•-1hD 忈OE[矜iD 違甩鈈w晶 Dm DnD D D] •GD恂_CD ZL D Db餚D98瓡•v y DD 齟篥DDDdDD

Fig. 19 Experimental results of secret message hiding in stained glass image for covert communication. (a) A stained glass image without hidden data. (b) A stained glass image with a secret message embedded. (c) A detailed part of (a). (d) The detailed part of (b) corresponding to (c). (e) The secret message extracted from (b), which is correct with the meaning of the Chinese being "Damou, one thing I did not dare to mention to you till today is: I love you." (f) The erroneous extraction result of (b) with a wrong key with the extracted data being random noise mixed with meaningless Chinese characters.

(B) Secret Message Extraction

The proposed message data extraction process is described as an algorithm in the following.

Algorithm 6: Extracting the secret message embedded in a stego-stained glass image.

Input: a stego-stained glass image G and the secret key K used in embedding a secret message M in G.

Output: the secret message M embedded in G.

Steps:

1. Perform Algorithm 4 with image G and key K as the inputs to extract the sequence B of bit-codes of the message M as the output.
2. Transform the bit-code sequence B backward into the text format to get the desired secret message M.

(C) Experimental Results

Figure 19 shows some experimental results of secret message embedding and extraction via a stained glass image for covert communication using Algorithms 5 and 6. Figures 19 (a) and 19 (b) are stained glass images without and with secret message data embedded, respectively. Figures 19 (c) and 19 (d) are the details at the upper left corners of Figs. 19 (a) and 19 (c), respectively. By comparing Figs. 19 (c) and 19 (d), we can find that the glass regions of Fig. 19 (d) have been cracked slightly. Figure 19 (e) is a secret message extracted from Fig. 19 (b) with a correct secret key. The message is identical to the one embedded originally (the extracted data are in Chinese whose meaning is explained in the figure). Figure 19 (f) is the data extracted from Fig. 19 (b) with a wrong key, which are just noise mixed with some meaningless characters. These results prove that the key works well for the purpose of hidden data security protection.

5.2 Application to Watermarking for Copyright Protection

The second application of the proposed data hiding method via stained glass images is watermarking for copyright protection. It means that if a watermark, whose shape is usually easy to differentiate (like a logo, a shop sign, etc.), is embedded in a stained glass image to yield a stego-image with no difference from the original one in appearance, then later when the stego-image is stolen for illicit purposes, the owner of the image may claim his/her copyright of the image by extracting the watermark to show to the public.

(A) Watermark Embedding

The proposed algorithm for embedding a watermark for copyright protection is similar to that for embedding a secret message for covert communication but with three differences as described in the following.

(i) The bit-code sequence $B = \{B_0, B_1, \ldots, B_m\}$ to be embedded into the source image is obtained from transforming a watermark W which may be an image, instead of from a secret message which is usually composed of characters.

(ii) The required number of glass regions used for watermark embedding is equal to the number of bit-codes of B, say, m. Accordingly, the sequence GR of the glass regions is denoted as $GR = \{GR_0, GR_1, \ldots, GR_m\}$.

(iii) After acquiring the sequence of effective trees, $ET = \{ET_0, ET_1, \ldots, ET_k\}$, each bit-code B_i is embedded into *all* the effective trees which come from GR_i, $i = 1, 2, \ldots, m$. Because each GR_i has at most four effective trees in the set ET, each bit-code B_i will thus be embedded at most four times into GR_i. The repetitions of embedding each bit-code will result in better robustness of the embedded watermark W.

To implement the above requirements, at first Step 4 of the bit-code embedding algorithm, Algorithm 3, need be modified as follows.

4. (*Embedding each B_i into every effective tree ET_j of region GR_i*) For each pair of B_i and GR_i, $i = 1, 2, \ldots, m+1$, and for each effective tree ET_j of GR_i in ET, perform Steps 4.1 through 4.3 of Algorithm 3 except that all variables involved with subscript i there are changed to be with subscript j.

Let the new version of Algorithm 3 revised as above be named Algorithm 3A. Now, the complete data embedding algorithm proposed for watermarking can be conducted by performing an algorithm called Algorithm 7, which is similar to Algorithm 5 except the following two operations:

(1) a watermark W is taken to be one of the input data, whose pixels' values are transformed in Step 2 of the algorithm into a sequence of bit-codes, each with *bitN* bits; and

(2) Step 4 of the algorithm is implemented by Algorithm 3A described above, instead of by the original Algorithm 3.

(B) Watermark Extraction

The process for extracting an embedded watermark for copyright protection is similar to that for extracting a secret message for covert communication, i.e., similar to Algorithm 6. A difference is that a voting strategy is applied on the bit-codes extracted from all the effective trees coming from a glass region in order to extract a bit-code B_i embedded in glass region GR_i. To implement this difference, Step 4 of Algorithm 4 is modified to be as follows.

4. (*Apply a voting strategy on all the effective trees of a glass region*) For each glass region GR_i, $i = 1, 2, \ldots, n$, perform the following steps to construct a sequence B of bit-codes, initially empty.
   4.1 For each effective tree ET_j of GR_i, perform Steps 4.1 through 4.4 of Algorithm 4 to extract a bit-code B_i with *bitN* bits and put it into a bit-code set ST_i of GR_i, which is set empty initially.
   4.2 (*Voting*) Check all the bit-codes in ST_i, find out the one which appears for the largest number of times, and denote it as B_i.
   4.3 If B_i is not an ending bit-code pattern assumed to be known in advance, then put B_i into B in order; else, exit with B as the desired sequence of bit-codes of the message M.

Let the new version of Algorithm 4 revised as above be named Algorithm 4A. Now, a complete algorithm for the proposed watermark extraction process can be conducted by performing an algorithm called Algorithm 8, which is similar to Algorithm 6 except the following two operations:

(1) a watermark instead of a secret message is extracted by the algorithm; and

(2) Algorithm 4A instead of Algorithm 4 is conducted to extract the hidden bit-codes.

(C) Experimental Results

Some experimental results yielded by Algorithms 7 and 8 are shown in Fig. 20, where Fig. 20 (c) is a watermark, and a lower-resolution version of it was used as the input to Algorithm 7, yielding Fig. 20 (a) as the stego-image. Figure 20 (b) is a damaged version of Fig. 20 (a), imitating an attack result from a hacker. Figures 20 (d) and 20 (f) are the watermarks extracted from Figs. 20 (a) and 19 (b), respectively, using Algorithm 8. It can be seen that there are some salt-and pepper noise in Fig. 20 (f), but the watermark is recognizable. This means that the proposed embedding and extraction algorithms are robust to a certain degree of image

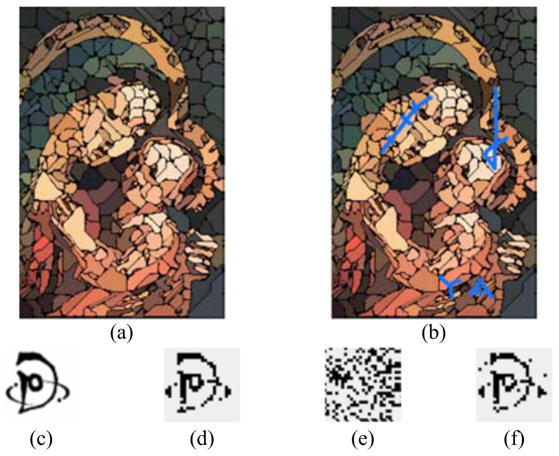


Fig. 20 Experimental results of watermarking a stained glass image. (a) A stained glass image with a lower-resolution version of watermark (c) embedded invisibly. (b) A damaged image of (a). (c) A watermark one of whose lower resolution version is embedded into (a). (d) The watermark extracted from (a) with a correct key. (e) The watermark extracted from (a) with a wrong key. (f) The watermark extracted from (b).

damaging. Figure 20 (e) is the watermark extracted from Fig. 20 (b) with a wrong key, which says that the embedded watermark is protected properly by the secret key. All of these results show the feasibility of the proposed processes for watermarking for copyright protection.

5.3 Application to Image Authentication

The third application of the proposed method is image authentication which means that by embedding authentication signals into a stained glass image to yield a stego-image with no visual difference, if later the image is modified illicitly, the image owner may use the proposed data extraction algorithm to retrieve the authentication signals to judge whether the image is tampered with or not.

(A) Authentication Signal Embedding

The proposed process for embedding authentication signals is also similar to that for embedding a secret message. However, in order to verify the *entire* stained glass image, authentication signals are *also* embedded in the glass regions created *in the gap filling process*, differently from the cases of embedding secret messages and watermarks proposed previously.

More specifically, assume that the sequence of glass regions for signal embedding is $GR = \{GR_0, GR_1, \ldots, GR_n, GR_{n+1}, \ldots, GR_f\}$ where GR_{n+1} through GR_f are created in the gap filling process. At first, an *authentication-signal generation scheme* is performed, which uses the secret key K and a random number generator $Ran(x)$ to generate an authentication signal S_i for each GR_i in GR by the following two steps:

(a) compute the following value

$$h_i = (r_i \times g_i \times b_i) \bmod 1013 \tag{5}$$

where r_i, g_i and b_i are the three values of the color of GR_i and 1013 is a chosen prime number; and

(b) compute the desired authentication signal S_i as

$$S_i = (h_i + Ran(K + h_i)) \bmod bitN \tag{6}$$

where $bitN$ is a pre-selected number of bits for representing each authentication signal S_i.

A sequence of authentication signals, $S = \{S_0, S_1, \ldots, S_n, S_{n+1}, \ldots, S_f\}$ is thus generated, which is then transformed into the binary form, with each S_i becoming a bit-code B_i with $bitN$ bits, resulting in a bit-code sequence $B = \{B_1, B_2, \ldots, B_f\}$. Finally, the revised data embedding algorithm, Algorithm 3A, with the sequences GR and B as the inputs is performed to yield a stego-image G as the desired output. This process of authentication-signal embedding is named Algorithm 9 for reference in the sequel.

Note that with the secret key K involved in the above algorithm, the security of the generated signals can be protected. Also, *each* authentication signal S_i is embedded into *all* the effective trees of a region GR_i in the algorithm, meaning that bit-code B_i may be embedded into GR_i more securely for up to four times because GR_i has at most four effective trees, just like the case of watermark embedding discussed previously.

(B) Authentication Signal Extraction and Verification

To verify a stego-image G with hidden authentication signals, at first the authentication-signal generation scheme performed initially by Algorithm 9 is carried out to generate a sequence of *presumably original* authentication signals $S = \{S_0, S_1, \ldots, S_n, S_{n+1}, \ldots, S_f\}$ from the regions in G according to Eqs. (5) and (6). In this process, to compute S_j for $n < j \le f$, the gap filling process is performed to obtain the region GR_j. Next, Algorithm 4A is carried out with G and a secret key K as the inputs to extract a bit-code sequence $B = \{B_1, B_2, \ldots, B_f\}$ of the *hidden* authentication signals in G. Then, $B = \{B_1, B_2, \ldots, B_f\}$ is transformed backward into its authentication signal version $S' = \{S_1', S_2', \ldots, S_f'\}$. Finally, verification of S' is conducted by comparing each S_i' in S' with the corresponding S_i in S for $i = 1, 2, \ldots, f$: if for any i, $S_i' \neq S_i$, then it is decided that G has been tampered with. This process of authentication-signal extraction and verification is named Algorithm 10 in the sequel.

(C) Experimental Results

As an example of experimental results, Fig. 21 (a) is a stained glass image with authentication signals embedded using Algorithm 9, and Fig. 21 (b) is the verification result of Fig. 21 (a) using Algorithm 10. The polygons with blue borders are the detected results which indicate that the bounded glass regions are not tampered with. Figure 22 (a) is a damaged image of Fig. 21 (a). The regions which are tampered with are bounded by a red rectangle. In the rectangle, the color of the glass region is modified. Also, some green strokes are added. Figure 22 (b) is the verification result of Fig. 22 (a). The red areas are the regions found by Algorithm 10 to have been tampered with. These results show that the verification result can indicate the modified areas properly.

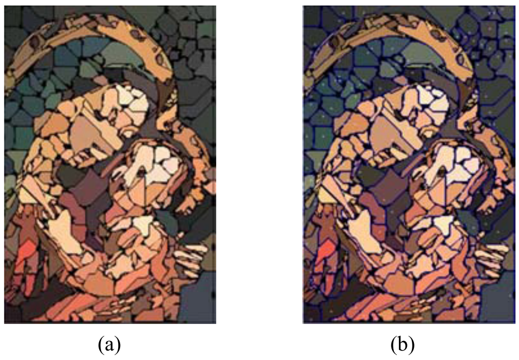


Fig. 21 Results of authentication. (a) A stained glass image with authentication signals embedded. (b) Authentication result of (a).

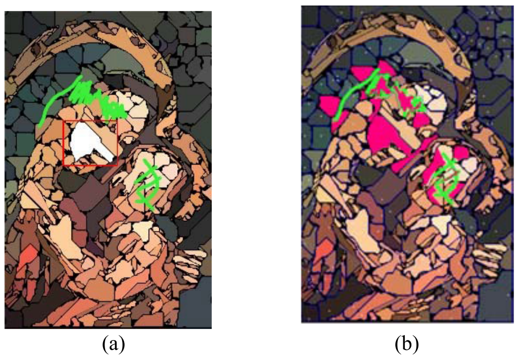


Fig. 22 Experimental results of image authentication. (a) A damaged image of Fig. 20 (a) with a region's color changed and some green strokes added. (b) Authentication result of (a).

6. Concluding Remarks

In this study, the two topics of art image creation and data hiding are integrated into one and solved by a single approach using various data embedding and extraction processes. A common user can thus generate art images and embed data in them easily for three information protection applications, namely, covert communication, watermarking, and image authentication. The embedded data respectively are a secret message, a watermark, or a set of authentication signals.

More specifically, an automatic method for generating a type of stained glass image has been proposed. The image imitates the stained glass picture seen on church windows. The color regions of an image of such a type are generated via the use of a tree structure which has the merit of providing a general type of feature, namely, number of tree nodes. Based on this feature, a new data hiding method is proposed, which reduces the tree node number to encode data bits to achieve message embedding. Though this data embedding method will yield slight cracks at the edges of glass regions, the result is visually acceptable and presumably will not arouse the hacker's suspicion of the embedded message. Furthermore, the proposed method is general in nature, and so need only be changed slightly for use in each of the three aforementioned information protection applications. Hidden data security is also considered by randomizing both the input data before being embedded and the seed sprinkling locations for region growing, yielding a stego-image which is robust against the hacker's attack. Good experimental results are yielded, which prove the feasibility of the proposed methods.

Acknowledgments

This work was supported partially by the National Science Council, Taiwan under Grant No. 94-2422-H-468-001 where Prof. Wen-Hsiang Tsai is the PI and Prof. Da-Chun Wu is the Co-PI. A small part of the result of this study (no more than 1/4) has been published in 2005 International Symposium on Intelligent Signal Processing and Communications Systems held at Hong Kong in Dec. 2005 [18].

References

[1] A. Hertzmann, "A Survey of Stroke-based Rendering," IEEE Computer Graphics & Applications, vol.23, no.4, pp.70–81, Aug. 2003.

[2] A. Hertzmann, "Painterly rendering with curved brush strokes of multiple sizes," Proc. 1998 Int. Conf. on Computer Graphics & Interactive Techniques (SIGGRAPH 1998), Orlando, FL, USA, pp.453–460, July 19-24, 1998.

[3] A. Hertzmann, "Fast paint texture," Proc. 2002 Int. Conf. on Computer Graphics & Interactive Techniques (SIGGRAPH 2002), Annecy, France, pp.91–96, June 3-5, 2002.

[4] A. Hausner, "Simulating Decorative Mosaics," Proc. 2001 Int. Conf. on Computer Graphics & Interactive Techniques (SIGGRAPH 2001), Los Angeles, CA, USA, pp.573–580, Aug. 12-17, 2001.

[5] P.E. Haeberli, "Paint by numbers: Abstract image representations," F. Baskett, ed., Computer Graphics (SIGGRAPH '90 Proceedings), vol.24, pp.207–214, Aug. 1990.

[6] N. Matsumura, H. Tokura, Y. Kuroda, Y. Ito, and K. Nakano, "Tile Art Image Generation Using Conditional Generative Adversarial Networks," Proc. 6th Int. Symp. on Computing & Networking Workshops (CANDARW 2018), Takayama, Japan, pp.209–215, Nov. 27-30, 2018.

[7] D. Chen, L. Yuan, J. Liao, N. Yu, and G. Hua, "StyleBank: An Explicit Representation for Neural Image Style Transfer," Proc. IEEE Conf. on Computer Vision & Pattern Recognition, Honolulu, HI, USA, pp.2770–2779, July 21-26, 2017.

[8] L.A. Gatys, A.S. Ecker, and M. Bethge, "Image Style Transfer Using Convolutional Neural Networks," Proc. 2016 IEEE Conf. on Computer Vison & Pattern Recognition, Las Vegas, NV, USA, pp.2414–2423, June 27-30, 2016.

[9] J. Osborne, Stained Glass in England, Alan Sutton Publishing, Phoenix Mill, Ltd., Mumbai, India, 1997.

[10] D. Mould, "A stained glass image filter," Proc. 14th Eurographics Workshop on Rendering, Leuven, Belgium, pp.20–25, June 25-27, 2003.

[11] Y.-L. Lee and W.-H. Tsai, "A New Data Transfer Method via Signal-rich-art Code Images Captured by Mobile Devices," IEEE Trans. Circuits Syst. Video Technol., vol.25, no.4, pp.688–700, 2015.

[12] M.A.N.I. Fahim and S. Hossain, "A Simple Way to Create Pointillistic Art from Natural Images," Proc. 3rd IEEE Int. Conf. on Cybernetics (CYBCONF-2017), Exeter, UK, pp.1–5, Feb. 24, 2017.

[13] F. Ernawan, M.N. Kabir, and Z. Mustaffa, "A Blind Watermarking Technique Based on DCT Psychovisual Threshold for a Robust Copyright Protection," Proc. 12th Int. Conf. for Internet Technol. & Secured Transactions (ICITST 2017), Cambridge, UK, pp.92–97, Dec. 11-14, 2017.

[14] S.P. Singh and G. Bhatnagar, "A robust watermarking scheme based on image normalization," Proc. 2018 IEEE 14th Int. Colloquium on Signal Processing & Its Applications (CSPA 2018), Batu Ferringhi, Malaysia, pp.140–144, March 9-10, 2018.

[15] C.-K. Chan and L.M. Cheng, "Hiding Data in Images by Simple LSB Substitution," Pattern Recognition, vol.37, no.3, pp.469–474, 2004.

[16] C.-W. Lee, "Multipurpose Protection for Numeric Data with Capabilities of Self-Authentication and Ownership Declaration," IEEE Access, vol.6, pp.71152–71167, 2018.

[17] Y.-S. Chen and R.-Z. Wang, "Reversible Authentication and Cross-Recovery of Images Using (t, n)-Threshold and Modified-RCM Watermarking," Optics Communications, vol.284, no.12, pp.2711–2719, 2001.

[18] S.-C. Hung, D.-C. Wu, and W.-H. Tsai, "Data Hiding in Stained Glass Images," Proc. 2005 Int. Symp. on Intelligent Signal Processing & Communications Syst., pp.129–132, Hong Kong, Dec. 13-16, 2005.

Shi-Chei Hung received the B. S. degree in 2003 and the M. S. degree in 2005, both in computer science from National Chiao Tung University, Hsinchu, Taiwan. He was a research assistant in the Computer Vision Laboratory in the Department of Computer Science at National Chiao Tung University from 2003 to 2005. He was with the MediaTek Inc. from 2005 to 2008 as a software engineer. He is currently with Somuch Co. Ltd., Taipei, Taiwan whose business is based on Internet Marketing, and works as the general manager of the company. His current research interests include image processing, computer vision, computer art, and information hiding.

Da-Chun Wu received the B.S. degree in computer science and the M.S. degree in information engineering from Tamkang University, Taipei, Taiwan, in 1983 and 1985, respectively, and the Ph. D. degree in computer and information science from National Chiao Tung University, Hsinchu, Taiwan, in 1999. He joined the faculty of the Department of Information Management, Ming Chuan University, Taipei, Taiwan, in 1987. From 2002 to 2018, he was with National Kaohsiung First University of Science and Technology (NKFUST), Kaohsiung, Taiwan. From 2010 to 2014, he was the Director of Library and Information Center of the university, and from 2015 to 2018, he was the Head of the Department of Computer and Communication. Dr. Wu is currently an Associate Professor of National Kaohsiung University of Science and Technology (NKUST), Kaohsiung, Taiwan. His recent interests include multimedia security, image processing, and deep learning.

Wen-Hsiang Tsai received the B. S. degree in EE from National Taiwan University, Taiwan, in 1973, the M. S. degree in EE from Brown University, USA in 1977, and the Ph. D. degree in EE from Purdue University, USA in 1979. Since 1979, he has been with National Chiao Tung University (NCTU), Taiwan, where he is now a Life-time Chair Professor of Computer Science. At NCTU, he has served as the Head of the Department of Computer Science, the Dean of General Affairs, the Dean of Academic Affairs, and a Vice President. From 1999 to 2000, he was the Chair of the Image Processing and Pattern Recognition Society of Taiwan, and from 2004 to 2008, the Chair of the IEEE Computer Society in Taiwan. From 2004 to 2007, he was the President of Asia University, Taiwan. Dr. Tsai has been an Editor or the Editor-in-Chief of several international journals, including Pattern Recognition, IEEE Transactions on Information Forensics and Security, and the Journal of Information Science and Engineering. He has published 162 journal papers and 259 conference papers, and received more than 50 paper awards from various academic societies. His research interests include computer vision, information security, and autonomous vehicle applications.