# DATA HIDING IN STAINED GLASS IMAGES*

Shi-Chei Hung[1], Da-Chun Wu[2] and Wen-Hsiang Tsai[1, 3]

[1]National Chiao Tung University
[2]National Kaohsiung First University of Science and Technology
[3]Asia University

## ABSTRACT

New methods for automatic generation of stained glass images as well as for data hiding in such images are proposed. Glass regions in stained glass images are generated by randomly sprinkled seeds and a region growing technique. The number of nodes of a tree constructed for pixel grouping in a glass region is used for data hiding in a stained glass image. A complete process for secret communication using the above technique is also proposed. Experimental results show the feasibility of the proposed methods.

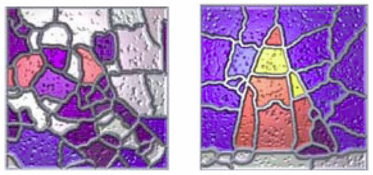
## 1. INTRODUCTION

*Stained glass windows* are composed of glass pieces, which are of different shapes, colors, and sizes. According to the investigations by Armitage[1] and Osborne[2], stained glass windows first appeared in the 7th century, had heyday at the 16th century, and still being built today. Fig. 1(a) shows the detail of a stained glass window. Although creating mosaic-effect images, composed of geometric elements such as small images, tiles, and glass pieces, is a new research topic in recent years, little attention has been paid to the historical successor to the mosaic, the stained glass image. Mould [3] proposed an algorithm for stained glass image creation. The method starts with an initial segmentation of a source image by an image processing system called EDISON proposed in [4, 5, 6]. Then, the segmented regions are manipulated and smoothed by erosion and dilation operators [7, 8]. Finally, a displacement map representing imperfections in the glass is created and leading is applied between the tile boundaries accordingly. Fig. 1(b) shows some results of Mould's method.

Researches on data hiding in images are mostly based on pixel-wise or block-wise operations and few image features are used. In this study, we propose a new method to create stained glass images that are suitable for data hiding. The method is based on the use of a new tree structure of image pixels. The proposed process for stained glass image creation is described in Section 2. The technique for data hiding by slight glass cracking is proposed in Sections 3 and 4. Some Experimental results are shown in Section 5. Section 6 concludes this paper with some suggestions for future works.

* This work was supported partially by the NSC Project, Advanced Technologies and Applications for Next Generation Information Networks (II), with Project No. NSC93-2752-E-009-006-PAE and partially by NSC Project No. NSC 94-2422-H-468-001.

(a)



(b)

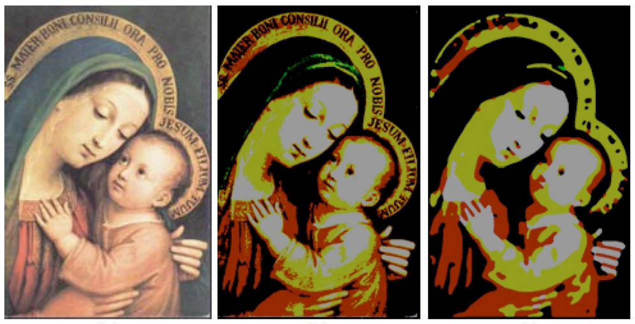
Fig. 1. Stained glass window and images. (a) Detail of The Crucifixion, St. James Church, Staveley, UK by Neil Ralley. (b) Some results of Mould's method.

## 2. PROPOSED STAINED GLASS IMAGE CREATION

In the proposed process for creating stained glass images which are suitable for data hiding, first we quantize each pixel value of an input color image into three bits, one bit per R, G, or B channel. Next, we filter off the noise appearing during quantization by a voting filter. The way we do for this is to accumulate, for each pixel, the numbers of pixel colors within a square which is centered at the pixel. Then we reset the color of that pixel to be the color which has the maximum accumulation value. After these two steps of image preprocessing, the input image is partitioned into several *color regions*. An example is shown in Fig. 2, in which (a) is an input image and (b) is the quantization result. Here the square size is 11×11. Fig. 2(c) is the result after filtering. We name Fig. 2(c) a *color region image*.

Continuing the image creation process, we sprinkle seeds on the color regions and apply a new region growing scheme on each seed. A random number generator is used for seed sprinkling.

In Section 2.1, a tree structure of glass regions and the above-mentioned region growing scheme will be described. By the region growing technique, we can grow glass regions from the sprinkled seeds. In the final step, we search for gaps which are of large areas among the created glass regions and fill up the gaps with extra glass regions. We name this process glass region gap filling, which will be discussed in Section 2.2. Finally, a stained glass image is created with glass colors specified by the seed positions and the input image.



(a) (b) (c)

Fig. 2. Preprocessing of a given image. (a) The original image. (b) A quantized image of (a). (c) A filtered image of (b).

## 2.1. Region growing process and tree structures of glass regions

In this section, we describe the proposed tree structure of glass regions for region growing. The tree structure is shown in Fig. 3(a). A tree node is either an interior node or a leaf node. An interior node has child nodes and a leaf node does not. We classify the tree nodes into two types, *succeeding node* and *expanding node*. Expanding nodes are the first or last nodes in a node level. In Fig. 3, we indicate expanding nodes by thicker borders and succeeding nodes by thinner borders. As shown in Fig. 3(a), each expanding node has at most two child nodes, one being an expanding node and the other a succeeding node. On the other hand, a succeeding node has at most a child node. And the child node is also a succeeding node.

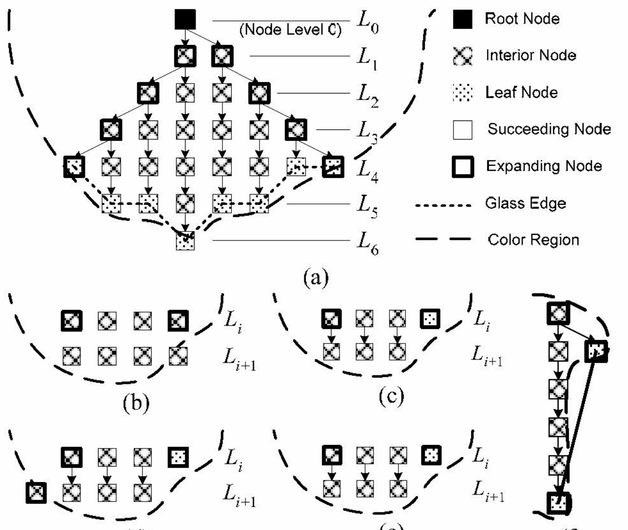


Fig. 3. Tree structure of glass regions. (a) A tree of a glass region, (b), (c), (d) and (e) examples of Steps 2 and 3 in Algorithm 1.

### 2.1.1. Process of growing for glass region

The proposed region growing process starts by rooting four trees in each seed, which are denoted by *Nt*, *Et*, *St*, and *Wt*, respectively. The four trees are then expanded in four directions, namely, north, east, south, and west, respectively, by a tree growing process to be described in Section 2.1.2. All the trees of all the sprinkled seeds are grown simultaneously and one level after another. After the tree growing processes of the four trees terminate, the leaf nodes of the four trees are linked to form the boundary of the associated glass region.

### 2.1.2. Process of tree growing

The proposed tree growing process is illustrated in Algorithm 1.

```
**Algorithm 1: tree growing.**
Input: a tree root $R$.
Output: a tree, $GRT$, which is grown from $R$.
Steps:
Step 1  Add two expanding nodes in node level $L_1$ as the child
        nodes of $R$. Set the states of them as interior nodes.
Step 2  For each node level $L_i$, where $i \geq 2$, perform the following steps:
    2.1 For each interior node, $N_{int}$ , in $L_i$, generate a child
        node which is interior and succeeding. Denote it by $C_{is}$.
        2.1.1 Check whether $C_{is}$ is out of bound by the following way.
        2.1.2 If $N_{int}$ and $C_{is}$ are not in the same color region,
              regard $C_{is}$ as out of bound and
            A. discard $C_{is}$ from $N_{int}$;
            B. trasfer $N_{int}$ from an interior node to a leaf node.
        2.1.3 If the depth of $C_{is}$ is greater than the depth of
              its neighboring sibling, then
            A. discard $C_{is}$ from $N_{int}$;
            B. trasfer $N_{int}$ from an interior node to a leaf node.
    2.2 For each interior node $N_{intE}$ which is also an
        expanding node, generate diagonally a child node
        which is interior and expanding. Denote it by $C_{ie}$.
        2.2.1 Check whether $C_{ie}$ is out of bound by the following way.
        2.2.2 If $N_{intE}$ and $C_{ie}$ are not in the same color region,
              regard $C_{ie}$ as out of bound and discard $C_{ie}$ from $N_{intE}$.
        2.2.3 If the depth of $C_{ie}$ is greater than the depth of
              its neighboring sibling, then discard $C_{ie}$ from $N_{intE}$.
    2.3 Check whether the growing process is converged by
        the following way:
        if there are child nodes generated in Steps 2.1 and 2.2, then
        2.3.1 regard $GRT$ as non-converged, accumulate the
              value of $i$ by 1, and go back to the beginning of
              Step 2; otherwise,
        2.3.2 regard $GRT$ as converged, output $GRT$, and
              terminate the growing process.
```

Figs. 3(b) and (c) show an example of Step 2.1.1. As shown in Fig. 3(b), four succeeding nodes are produced in $L_{i+1}$. The rightmost one is located in a color region different from where its parent node is located. So, we regard this node as out of bound, discard it from its parent node, and transfer its parent node from an interior node to a leaf node. Figs. 3(d) and (e) is an example of Step 2.2.1. As shown in Fig. 3(d), only one node is produced in $L_{i+1}$. Because the rightmost expanding node in $L_i$ has been transferred into a leaf node in Step 2.1.1, we will not extend this node any more. As shown in Fig. 3(e), the generated child node is located in a color region different from where its parent node is located, so we discard it without transferring its parent node to a leaf node.

In this algorithm, Steps 2.1.2 and 2.2.2 are used for keeping the depth differences of neighboring leaf nodes less than or equal to one. The reason for doing that is shown in Fig. 3(f). In Fig. 3(f), the depth difference of the neighboring two leaf nodes is four. The solid line in Fig. 3(f) is the edge defined by the two leaf nodes. We can see that this edge overlaps another color region. It will result in overlapping glass regions when the growing process terminates. That is why we have to keep the depth differences of neighboring leaf nodes to be smaller than or equal to one by Steps 2.1.2 and 2.2.2. Finally, in Step 2.3, we decide whether to terminate the growing process or not. If the process terminates, the derived tree will be part of a glass region in the stained glass image we created.

### 2.2. Process of glass region gap filling

At the beginning of the proposed glass growing process, we sprinkle seeds as tree roots randomly. It will result in more than one seed sprinkled in a color region. On the contrary, it might also happen that no seed is sprinkled in a color region, causing some gaps among the glass regions. So, we have to perform a further step to fill these gaps. We name this step the gap filling process. Initially, we scan the tree map. If there is no tree node within a square range of size 8x8 pixels, we put an additional seed at the center of that square and apply the proposed glass region growing process on it. Fig. 4 is the final stained glass image of the input image of Fig. 2(a).

## 3. CONCEPTS BEHIND PROPOSED DATA HIDING TECHNIQUE

The feature we utilize for data hiding in stained glass images is the number of nodes in a tree of a glass region. By removing the deepest nodes, we can hide data in a stained glass image. However, this method will result in cracks at the edges or corners of glass regions, though still acceptable. As shown in Fig.

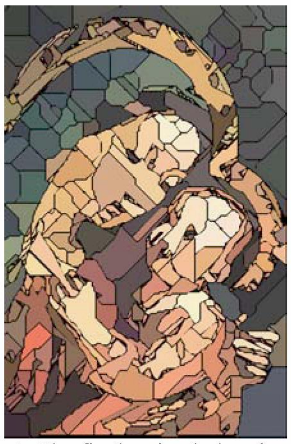


Fig. 4. The final stained glass image.

5(a), we remove the two deepest nodes in tree *Nt* of the glass region. Fig. 5(b) is the result of removing the two nodes. We can see the resulting cracks in the glass region where the nodes are removed. There are four trees contained in a glass region, but not all the four trees can be used for data hiding. The number of nodes of a tree must be large enough so that data can be embedded by removing tree nodes.

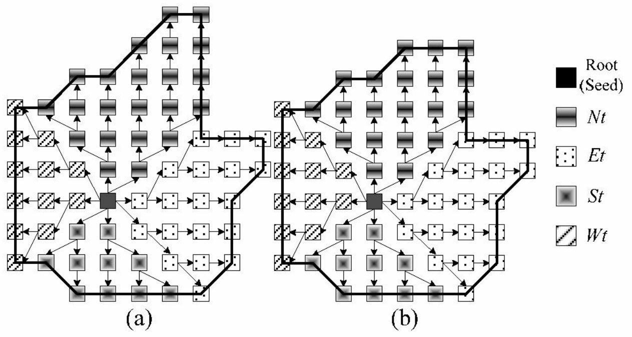


Fig. 5. An example of removing nodes for data hiding. (a) A glass region. (b) A glass region with data embedded.

## 4. DATA HIDING BY GLASS BOUNDARY CRACKING

The core concept of data hiding by modification of the number of tree nodes is illustrated in Fig. 6. Initially, we compute the remainder, *REM*, of dividing the number of tree nodes, *TNN*, by a divisor, *DIV*. The value of *REM* will thus range between 0 and *DIV*−1 as shown in the following formula:

$$REM = TNN \bmod DIV,\ 0 \leqq REM < DIV-1. \qquad (1)$$

Then, we divide the range of *REM* into several sub-ranges. Each sub-range is regarded to represent a specific bit code. In the case of Fig. 6, the bit code contains two bits, so we divide the range into four sub-ranges which represent the bit codes 00, 01, 10, and 11, respectively. In other words, the number of sub-ranges, *NSR*, is computed by the following formula:

$$NSR = 2^{bitN}, \qquad (2)$$

where *bitN* is the number of bits we want to embed in an effective tree. Furthermore, because each sub-range includes several *REM* values, tolerance of detection errors can be achieved. Take Fig. 6 as an example. Each sub-range includes three *REM* values. No matter what the value of *REM* is (zero, one, or two), it means that we have embedded bit code 00 in the associated tree. The three rest sub-ranges may be deduced by analogy. Finally, the value of *DIV* can be computed as follows:

$$DIV = SSR \times NSR, \qquad (3)$$

where *SSR* is the number of *REM* values spanned by a sub-range. So, the value of *DIV* in Fig. 6 is 3 × 4 = 12. After deciding the values of *bitN* and *SSR*, the data embedding and extraction methods can be described as follows.

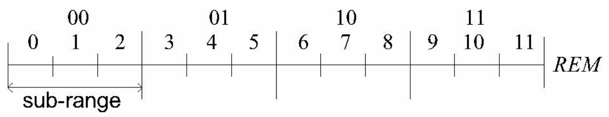


Fig. 6. Illustration of concept of data hiding in stained glass images.

### 4.1. Data embedding and extraction

If we want to embed a bit code into an effective tree, we adjust the value of *REM* to be the center of the sub-range of *REM* which represents the bit code we want to embed. Take Fig. 6 as an example. Assume that the value of *REM* is 9 and the bit code we want to embed is 01. The sub-range representing bit code 01 is the second one which centers on the *REM* value of 4. So the number *NNR* of nodes to remove from the tree for embedding bit code 01 is computed as follows:

$$NNR = | REM - \text{central value of target sub-range}| = |9 - 4| = 5.$$

By removing the deepest five nodes from the target tree, bit code 01 can be embedded.

The process of extracting data from a tree is simply to compute the value of *REM* and find the sub-range into which it falls. The corresponding bit code is the one we want to extract.

## 5. EXPERIMENTAL RESULTS

Fig. 7 shows some experimental results of data hiding in a stained glass image. Figs. 7(a) and (b) are stained glass images without and with data embedded, respectively. Figs. 7(c) and (d) are the details of the upper left corners of Figs. 7(a) and (c), respectively. By comparing Figs. 7(c) and (d), we can find that the glass regions of (d) are cracked slightly. Fig. 7(e) is a secret message extracted from Fig. 7(b). The message is identical to the one we embedded. Fig. 7(f) is the secret message extracted from Fig. 7(b) with a wrong key. We can see that the text shown in Fig. 7(f) is disordered and meaningless. This proves that the key used in the proposed creation process really works.

## 6. CONCLUSIONS AND FUTURE WORKS

In this study, we have proposed methods for stained glass image creation and data hiding. These two topics are integrated into one which is then solved by a single approach in the proposed methods, so that a common user can easily generate stained glass images and embed data in them. We utilize the data hiding method for embedding secret messages to achieve the goal of secret communication. However, it may also be applied to embed watermarks and authentication signals which can help us to achieve the goal of copyright protection and image verification, respectively. According to our research, we can claim that if there is an image feature of an art image that can be modified and detected, there will be a corresponding data hiding technique.

## 7. REFERENCES

[1] Armitage, E. L., "Stained Glass: History, Technology, and Practice," *Newton*, Charles T. Branford Company, 1959.

[2] Osborne, J., *Stained Glass in England*. Alan Sutton Publishing, Phoenix Mill, 1997.

[3] Mould, D., "A Stained Glass Image Filter," *Proceedings of the 14th Eurographics workshop on Rendering*, Leuven, Belgium, 2003, pp. 20-25.

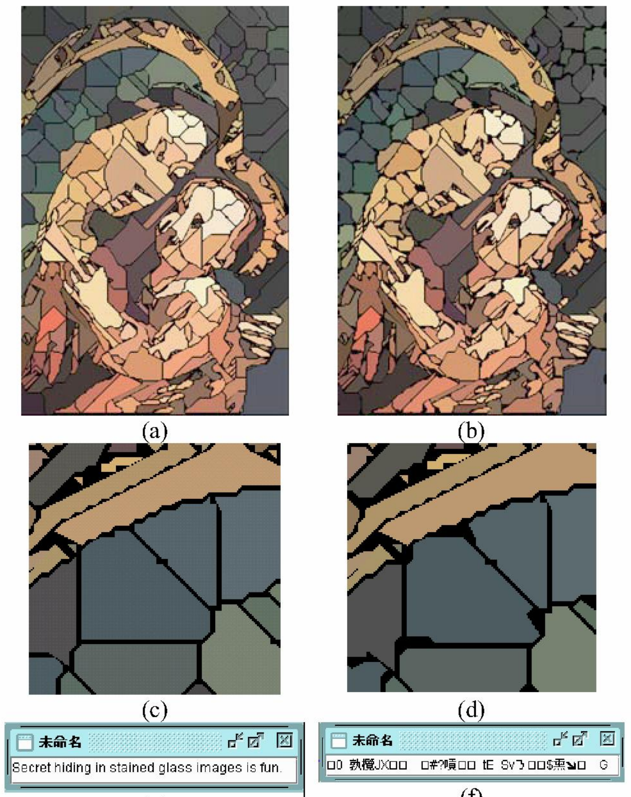
[4] Christoudias, C., Georgescu, B., and Meer, P., "Synergism in low-level vision," *International Conference on Pattern Recognition 4*, 16 Aug. 2002, pp. 150-155.

[5] Comanicu, D. and Meer P., "Mean Shift: A Robust Approach Toward Feature Space Analysis," *IEEE Trans. Pattern Anal. Machine Intell.*, *24*, 4 May 2002, pp. 603-619.

[6] Meer, P. and Georgescu, B., "Edge Detection with Embedded Confidence," *IEEE Trans. Pattern Anal. Machine Intell.*, *23*, 12 Dec. 2001, pp. 1351-1365.

[7] Arthur, R. and Weeks, J., "Fundamentals of Electronic Image Processing," *SPIE Optical Engineering Press*, Bellingham, 1996.

[8] Shapiro L. and Stockman G. *Computer Vision*. Prentice-Hall, Upper Saddle River, 2001.



Secret hiding in stained glass images is fun.

□0 孰欖JX□□ □#?嗔□□ tE Sv҄ □□$悪↘□ G

Fig. 7. Experimental results of data hiding in stained glass image. (a) A stained glass image without hidden data. (b) (a) with secret message embedded. (c) Details of (a). (d) Details of (b). (e) Secret message extracted from (b). (f) Extraction result of (b) with a wrong key.