

國立交通大學

資訊科學與工程研究所

碩士論文

供資訊隱藏應用之大量資料植入與搜尋技術之
研究

A Study on Large-volume Data Embedding and Search
Techniques for Information Hiding Applications

研究生：賴宣宏

指導教授：蔡文祥 教授

中華民國九十六年六月

供資訊隱藏應用之大量資料植入與搜尋技術之研究
A Study on Large-volume Data Embedding and Search Techniques for
Information Hiding Applications


研 究 生：賴宣宏

Student：Hsuang-Huang Lai

指 導 教 授：蔡文祥

Advisor：Wen-Hsiang Tsai

國 立 交 通 大 學
資 訊 科 學 與 工 程 研 究 所
碩 士 論 文



A Thesis
Submitted to Institute of Multimedia Engineering
College of Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
in
Computer Science
June 2007
Hsinchu, Taiwan, Republic of China

中華民國九十六年六月

供資訊隱藏應用之大量資料植入與搜尋技術之研究

研究生：賴宣宏

指導教授：蔡文祥 博士

國立交通大學資訊科學與工程研究所

摘要

隨著電腦科技的進步與網路的普及，越來越多的資料可在網路上快速且便利的傳送。本論文首先提出在 GIF 檔案上的一個大量且無損的資訊隱藏法。在這方法中，我們複製影像調色盤中出現頻率高的顏色到調色盤中沒用到的空格中，並且利用被複製的顏色來藏入資訊。實驗發現，用出現頻率越高的顏色來藏入越多位元，整體隱藏量會提高許多。接著，我們提出在 PNG 影像上一個可以在網頁上植入祕密資訊的方法。此方法的總隱藏量取決於前景色 PNG 的透明度通道所使用的隱藏位元數。隱藏完之後，我們利用提出的方法來調整前景圖和背景圖的灰階強度來減低資訊隱藏後所造成的雜訊。最後，我們提出兩個可在 BMP 影像資料庫中快速找尋所需影像的方法。一個是非區塊層次(non-block-level)的方法，在此方法中我們把註解資訊循序的藏到影像裡；另一個則是區塊層次(block-level)的方法，在此方法中我們先把影像分成區塊後並分別把註解資訊藏到區塊裡。實驗結果顯現出這些方法的靈活度與實用性。

A Study on Large-volume Data Embedding and Search Techniques for Information Hiding Applications

Student: Hsuang-Hunag Lai Advisor: Prof. Wen-Hsiang Tsai

Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

ABSTRACT

With the advance of computer technologies and the popularity of the Internet, more and more data can be transmitted speedily and conveniently on public networks. In this study, first we propose a distortion-free and high-capacity data hiding method on GIF. This method duplicates colors of high frequencies to fill the unused color entries in the palette and uses the duplicated colors to hide secret message bits. It is found that more bits can be hidden by the method if colors with higher appearance frequencies are duplicated first. For PNG images, we propose a data hiding method for implanting secret messages in web pages. The hiding capability is achieved by changing the transparency values of the alpha channel of the pixels of the foreground PNG image. A scheme to adjust the intensity values of both foreground and background image pixels to reduce the artifacts caused by data hiding is also proposed. Finally, we propose two fast methods for searching desired BMP images in databases. One is a non-block-level method that hides comments in images sequentially, while the other is a block-level method that divides images into blocks in which comments are embedded. Good experimental results show the feasibility and applicability of the proposed methods.

ACKNOWLEDGEMENTS

I am in hearty appreciation of the continuous guidance, discussions, support, and encouragement received from my advisor, Dr. Wen-Hsiang Tsai, not only in the development of this thesis, but also in every aspect of his personal growth.

Thanks are due to Mr. Chih-Jen Wu, Mr. Kuan-Chieh Chen, Mr. Jian-Jhong Chen, Mr. Tsung-Chih Wang, Mr. Yi-Fan Chang, and Mrs. Kuan-Ting Chen for their valuable discussions, suggestions, and encouragement. Appreciation is also given to the colleagues of the Computer Vision Laboratory in the Institute of Computer Science and Engineering at National Chiao Tung University for their suggestions and help during my thesis study.

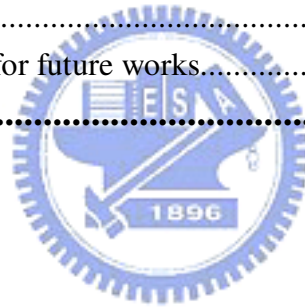
Finally, I also extend my profound thanks to my family for their lasting love, care, and encouragement. I dedicate this dissertation to my beloved parents.



CONTENTS

ABSTRACT	ii
ACKNOWLEDGEMENTS	iii
CONTENTS	iv
LIST OF FIGURES	vi
LIST OF TABLES	vii
Chapter 1 Introduction	1
1.1 Motivation	1
1.2 Survey of Related Studies	2
1.3 Overview of Proposed Method.....	4
1.3.1 Definitions of Terms	4
1.3.2 Brief Descriptions of Proposed Methods	5
1.3.2.1 Proposed Data Hiding Method for GIF (Graphics Interchange Format) Images	5
1.3.2.2 Proposed Data Hiding Method for PNG (Portable Network Graphics) Images.....	5
1.3.2.3 Proposed Fast Search Methods for Retrieving Desired Images fro Image Databases.....	6
1.4 Contributions	7
1.5 Thesis Organization	7
Chapter 2 A High-capacity and Distortion-free Data Hiding Method for GIF Images	8
2.1 Introduction	8
2.1.1 Usages of GIF Images	8
2.1.2 Properties of GIF Images	9
2.2 Proposed Data Hiding Method	10
2.2.1 Principle of Proposed Method	11
2.2.2 Data Embedding Process.....	14
2.2.3 Data Extraction Process	15
2.3 Experimental Results	16
2.4 Discussion and Summary	18
Chapter 3 Data Hiding by Stacking Up Two PNG Images	19
3.1 Introduction	19
3.1.1 Uses of PNG Images	19
3.1.2 Properties of PNG Images	20
3.2 Proposed Data Hiding Method	22

3.2.1	Proposed Idea	22
3.2.2	Data Embedding Process.....	24
3.2.3	Data Extraction Process	25
3.3	Experimental Results	25
3.4	Discussion and Summary	28
Chapter 4	Fast Search Methods via Hidden Messages in Images	31
4.1	Introduction	31
4.1.1	Need of Fast Search of Desired Images	31
4.1.2	Possible Techniques for Speeding Up Searches	32
4.2	Proposed Search Method	32
4.2.1	Proposed Ideas	32
4.2.2	Non-block-Level Search Method.....	35
4.2.3	Block-Level Search Method.....	36
4.3	Experimental Results	38
4.4	Discussion and Summary	41
Chapter 5	Conclusions and Suggestions for Future Works	42
5.1	Conclusions	42
5.2	Suggestions for future works.....	42
References	44



LIST OF FIGURES

Figure 3.1 An experimental result of the proposed method: (a) the background image; (b) the foreground image; (c) the piled-up cover image; (d) the stego-image without modifications. (e) the stego-image with modifications. (continued) 28

Figure 3.2 An experimental result of the proposed method: (a) the background image; (b) the foreground image; (c) the piled-up cover image; (d) the stego-image without modifications. (e) the stego-image with modifications. (continued) 30

Figure 4.1 The data structure of the non-block-level method. 34

Figure 4.2 The data structure of the block-level method. 34

Figure 4.3 Result of hiding comment words. (a) A cover image bee.bmp. (b) The resulting stego-image..... 35

Figure 4.4 Result of hiding comment words. (a) A cover image beauty.bmp. (b) The resulting stego-image..... 37

Figure 4.5 The particular image, Lena.bmp. 39

Figure 4.6 Image database used in the experiment. (a) Some images in the database and the first two images are Lena1.bmp and Lena2.bmp with comments “Lena is a model” and “The girl who lives in the U.S. and studies computer science is called Lena Wang.”, respectively. (b) Two desired images found in 0.04842 and 0.1046 seconds, respectively. (continued) 41

LIST OF TABLES

Table 2.1 Characteristics of tested images and their hiding capacity	16
Table 2.2 The comparison of cover images and stego-images.....	17
Table 3.1 The structure of IHDR chunk.....	21



Chpater 1

Introduction

1.1 Motivation

With the popularity of computer networks, more information is transmitted speedily through the public network environment with the risk of being detected or intercepted illegally. To avoid this, the idea of hiding data in images had been suggested. Various existing data hiding methods focus on different aspects. Some of them focus on hiding great amounts of data since more and more data are transmitted via the Internet. To develop a high-capacity data hiding method is necessary for such applications and is the first goal of this study.

Some other methods focus on raising the quality of the resulting image with hidden data (called stego-image in the sequel) since in fields like military and medical applications this is an urgent issue. In these fields, the distortion produced after data hiding may result in great danger. A general may misjudge due to a building-like object in the picture, but the object is actually noise yielded by data embedding. This is probably the cause of the death of the whole army in a battle, the loss of a war, or even the ruin of a country. A doctor may make wrong diagnosis due to some unknowns in an X-ray or a CT (computer tomography) image. This may result in a wrong treatment of the patient and is really a serious problem. To develop a distortion-free data hiding method is the second goal of our study.

Nowadays, images can easily be made by digital cameras, scanners, or be available on the Internet. We sometimes may want to find out a desired image in discs

at once but fail to recall the name of it. To solve this dilemma, we can hide the related information (we call such information a *comment* in the sequel) inside an image and build a searching mechanism based on that information. And this is the third goal of our study.

1.2 Survey of Related Studies

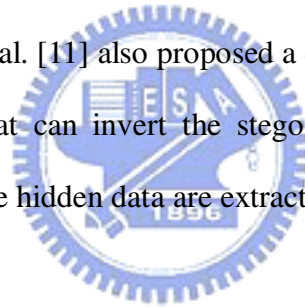
1.2.1 Review of Existing Data Hiding Techniques for Images

In recent years, several data hiding methods in images have been proposed and they can be roughly categorized into two groups: one includes spatial-domain methods and the other frequency-domain ones. In spatial-domain ones, secret data are directly embedded in the pixels of the cover image by replacing the least significant bits (LSBs) in it. On the contrary, in frequency-domain ones, the cover image is first transformed into the frequency domain, and then the secret data are embedded in the coefficients of the frequency domain.

One of the simplest ways of hiding secret messages in a cover image in the spatial-domain is to replace the LSBs of each image pixel. The LSB method was first proposed by Adelson [1] in 1990. Its advantages are fast and easy to implement, and being able to hide a great amount of information without causing perceptible distortion to the image. To increase the data hiding capacity, more LSBs may be altered to hide data but the expense is the quality degradation of images. It is common nowadays that two or more bits are used for embedding [2, 3].

Though LSB replacement methods are easy to implement as mentioned above, they are easy to detect by statistical analysis. An attacker can just apply simple signal

processing techniques to destroy the whole hidden data. The frequency-domain methods that hide data in significant areas of the cover image are much more robust against attacks, such as compression, resizing, and so forth, than the LSB methods. Nevertheless, they still remain imperceptible to the human visual system. Among the existing frequency-domain methods, one popular one is to use the discrete cosine transform (DCT) coefficients [4]-[7] for data hiding. In Yen and Tsai's [8] method, the cover image is first transformed into the frequency domain and then the DCT coefficient replacement method is utilized to accomplish the data hiding task. Another approach is the use of wavelet transforms [8]-[9]. Chang and Tsai [10] proposed an image data hiding based on the wavelet transform. Secret messages are embedded within the cover image by replacing the wavelet coefficients of the middle and high frequencies. Xuan and Zhu et al. [11] also proposed a data hiding method based on the integer wavelet transform that can invert the stego-image into the original image without any distortion after the hidden data are extracted.

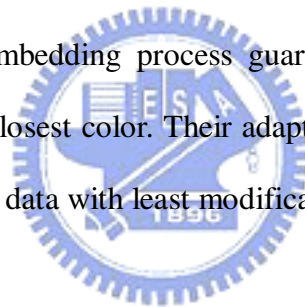


1.2.2 Review of Existing Data Hiding Techniques for Palette Images

For the topic of data hiding in palette images, either the palette or the image data can be used to hide secret data. A program called “Gifshuffle” was developed by Kwan [12] to embed data in GIF images. The principle is to permute the colors in the palette of the image in a specific order in accordance with the secret data. So there are totally $256!$ possible permutations of the 256 entries of the color palette. It means that at most $\log_2(256!)$ bits can be embedded into a GIF image. The advantage is that no perceptible distortion is introduced to the cover image. However, it has a fatal shortcoming that palette image software usually rearranges the order of the colors in

an image's palette so that the secret data will be dismissed when the image is being loaded and saved. Besides, a palette with randomly-ordered color entries might also be suspected as a stego-image. In some steganographic applications [13-16], by using the dithering method, the palette size can be doubled and thus utilized to choose appropriated colors to replace those of the pixels where data are embedded. But it was reported in [17] that if data hiding is achieved by manipulation of the color palette, hidden data is much likely detected by steganalysis.

Fridrich and Du [18] proposed a method that embeds data in palette images by first assigning a parity bit to each color in the palette and then adjusting the pixel index values in such a way that the parities of the new index values are equal to the message bits to be embedded. They also proposed an optimal parity assignment algorithm so that the data embedding process guarantees that an index is always replaced by the index of the closest color. Their adaptive method can be employed to conceal a moderate amount of data with least modification of pixel values.



1.3 Overview of Proposed Method

1.3.1 Definitions of Terms

The definitions of several terms used later are given first as follows.

1. Cover image: a cover image is an image into which a watermark signal is embedded.
2. Stego-image: a stego-image is an image that is produced by embedding a watermark signal into a cover image.
3. Embedding process: an embedding process is a process to embed data into an image.

4. Extraction process: an extraction process is a process to extract a watermark from a stego-image.

1.3.2 Brief Descriptions of Proposed Methods

1.3.2.1 Proposed Data Hiding Method for GIF (Graphics Interchange Format) Images

A steganographic method is proposed in this study, which exploits the use of GIF images' characteristics. First, a given GIF file is processed with some analyses to count the number of appearing colors and their appearing frequencies in that GIF image. Then the palette (consisting of squares) of it is modified by duplicating the high-frequency colors and put them into the palette's unused squares. Then the index array of the GIF image is modified according to the messages to be embedded. Since each of the colors used for embedding has more than one index, the changes of the indices after embedding do not result in any distortion. And since the hiding capacity is calculated by multiplying the number of bits a hiding color can hide and the appearing frequencies of the hiding colors, the more frequently the hiding colors appear, the more secret data the image can hide.

1.3.2.2 Proposed Data Hiding Method for PNG (Portable Network Graphics) Images

In this study, we propose a method to hide data in PNG images. First, we pile a PNG image on a gray-level one and then hide some secret data in the alpha channel (which determine the transparency of a PNG image) of it by hiding the data in its LSBs. Then a calculation is made to determine how many bits each byte of the

gray-level image can be used for hiding for the sake of reducing the resulting distortion.

1.3.2.3 Proposed Fast Search Methods for Retrieving Desired Images fro Image Databases

We proposed two fast search methods which can be used to find out the desired images efficiently. One applies both techniques of hashing and word-length counting while the other divides the image into several blocks and then conducts the hashing operation. And then the results of them are hidden altogether with the secret messages into the image. In another word, we sacrifice the hiding capacity for reaching faster search. With these preprocesses, the time for comparison of the keyword and the hidden messages in the search process can be reduced.

Some major contributions of the study are listed as follows.

- (1) A complete system for automatically creating personal talking cartoon faces is proposed.
- (2) A method for construction of 3D cartoon face models based on 2D cartoon face models is proposed.
- (3) A method for simulation of head tilting and turning using 3D rotation techniques is proposed.
- (4) Some methods for automatically gathering audio features for speech segmentation are proposed.
- (5) A method for simulation of the probabilistic head movements and basic emotions is proposed.
- (6) Several new applications are proposed and implemented by using the proposed system.

1.4 Contributions

Several contributions have been made in this study, as described as follows.

1. A steganographic method with distortion free capability is proposed for hiding large-volume secret messages in GIF images.
2. A steganographic method piling up two different kinds of images for hiding secret messages is proposed.
3. A fast search method for retrieving desired images from image databases is proposed. We directly use what are hidden inside the images and a key word for finding out the images we want. This saves spaces since no additional space is required to save those comments in images.

1.5 Thesis Organization



In the remainder of this thesis, the proposed high-capacity and distortion-free method for hiding data in GIF images is described in Chapter 2. In Chapter 3, the proposed method for hiding data in PNG images and gray-level images is described. In Chapter 4, a fast image search method via the hidden messages in images is described. Experimental results are shown at the end of each chapter. Finally, some conclusions and suggestions for future works appear in Chapter 5.

Chapter 2

A High-capacity and Distortion-free Data Hiding Method for GIF Images

2.1 Introduction

Nowadays, data hiding in images has become more and more popular and drawn lots of attention. This technique has been used in various areas, such as digital watermarking, secret transmission, secret sharing, and so on. Nevertheless, most of the existing data hiding methods introduce some noise after embedding. Though the noise is usually not imperceptible by human eyes, in some fields, such as military and medical ones, the quality of the stego-image is a critical issue. A malpractice will happen if there exist some unknown artifacts produced after hiding.

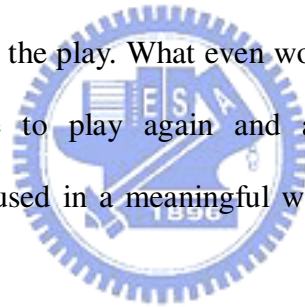
Since the GIF image will be used as the cover image for hiding data in this study, some properties and usages of the GIF image will be described in the rest of this section. In Section 2.2, the data hiding and extracting methods for GIF images will be described. In Section 2.3, some experimental results of applying the proposed methods will be illustrated. At last in Section 2.4, some discussions and the summary of the proposed method will be made.

2.1.1 Usages of GIF Images

GIF is the abbreviation of Graphics Interchange Format which was developed by CompuServe Inc. in 1987. Since it's widely supported by its portability and web browsers, it has become one of the most famous image formats on the World Wide

Web. GIFs are suitable for sharp-edged art that is of a limited number of colors. The most representative sharp-edged art is logo. The reason why GIFs are suitable for logos is that the lossless compression method used for creating the GIF image preserves sharp edges. This is an obvious advantage, as compared with JPEGs.

GIFs are also useful for short animations and low-resolution film clips. The GIF format allows a user to combine several GIFs into a single one to create animations. However, this has two drawbacks. One is that the GIF format applies no compression between the image frames of an animation so that if several larger GIFs are combined into one, a large-sized animation will be created. This contradicts one of the objectives of GIF development: being small in size. The other drawback is that no control interface exists, and the animation will play automatically, when downloaded, even if the user does not want the play. What even worse is that if looping is enabled, the animation will continue to play again and again. Actually the animation functionality of it is seldom used in a meaningful way and usually a disturbance to readers.



2.1.2 Properties of GIF Images

Each GIF image has a palette containing numerous squares and each of which has a color from the 24-bit RGB color space. The number of squares of a GIF image is a power of two and the maximum of it is 256. Besides the palette, a GIF image file also contains a two-dimensional index array in which each element is a number ranging from 0 to the size of the palette. Each pixel is associated with an index in the array and each index points to a color in the palette. So pixels with the same index values have the same color.

GIFs apply the LZW(Lempel Zev Welch) compression method that compresses an image without losing data and distorting the image itself. It is the method that best

compresses images with large fields of homogeneous colors. Less efficiency appears when a picture is of many colors and complex textures. It can be improved by reducing the number of colors in a GIF to be as small as possible. This is achieved by removing stray colors that are not required to represent the image, hence shrinking the size of it.

Another feature of the GIF file format is that the user is able to decide whether an image is interlaced or not. It means that if a GIF image is interlaced, supporting browsers display it in a low-to-high order of resolution, or if this is not the case, they do it in a raster order. This not only has the “fuzzy-to-sharp” effect that attracts the reader but gives the reader a preview of the full version of the image also. The feature is much more suitable for displaying larger GIF images since when displayed, a smaller interlaced one consumes more time than one without the interlacing feature. And generally speaking, the interlacing feature has little effect on the size of GIF images.

The last feature introduced here is free control of its transparency. This allows the users to pick colors in the palette of a GIF image to be transparent. Usually the background colors are selected for transparency when the image is to be displayed as a foreground. Unfortunately, if a color is chosen to be transparent, all of the pixels with that color will become transparent. So if a foreground pixel happens to have the same color as a background one, the foreground pixel will become transparent, which is usually undesired.

2.2 Proposed Data Hiding Method

2.2.1 Principle of Proposed Method

As mentioned above, the palette consists of several squares and each pixel has an index associated with a color square in the palette. We denote the “index-and-square” pair as (index, square). For example, we denote the color a pixel has as (5, square5) if and only if the index of the pixel is 5. So if two different squares are of the same RGB value, the pixels associated with either one of the squares will have the same color. That is, if square5=square6 (meaning that the two squares in the palette are of the same color), those pixels with index values 5 and 6 will have the same color when displayed. It means that if we change a pixel’s value from 5 to 6, no change will be introduced into the image. We can take advantage of this property to achieve high-capacity and distortion-free data hiding, as done in this study.

Plenty of 8-bit GIF images do not use all of the squares in the palette [18]. We name such squares *unused squares*. The principle of the proposed data hiding algorithm is *to duplicate colors appearing in used squares and fill them into the unused squares*. But not all of the used squares are duplicated; only those colors with high appearance frequencies will be duplicated and we call them “copy colors.”

Suppose that one of the copy colors is A and its index-and-square pair is (A, squareA) and the duplicated color is B and its index-and-square pair is (B, squareB). When embedding certain given secret data, we first conduct raster scanning and let the pixel currently being scanned be denoted as p . The scan is to check which pixel in the index array of a given cover image is of color value A. Then the data embedding rule proposed in this study goes as follows:

regard bit 0 being embedded at p if p has color A;

regard bit 1 being embedded at p if p has color B.

Since SquareB is duplicated from SquareA (so the colors in squares A and B are

identical), we know that the stego-image will appear to be completely the same as the cover image no matter how many As are modified to be Bs.

An example is given here. Suppose color A is in the i th square of the color palette, and let it be copied into the j th square of the palette and named color B. So we have two identical colors A and B in the palette with indices i and j , respectively. Now, to embed a bit 0 in the currently-scanned pixel p , we just associate the index i with p , and to embed 1, associate the index j with p .

The following section discusses how “high capacity” is achieved in this study. It is inspired from the previous section that if we duplicate a square more and more times, that copy color can hide more bits every time it appears. For example, if the copy color A is duplicated three times and the copy colors are named B, C, and D respectively. When embedding secret data, we can first find which pixel value is A and then embed secret bits as follows:

regard “00” being embedded in p if $p(x, y) = A$;

regard “01” being embedded in p if $p(x, y) = B$;

regard “10” being embedded in p if $p(x, y) = C$;

regard “11” being embedded in p if $p(x, y) = D$;

The same principle applies when a copy color is to be used to embed more bits. That is, if we duplicate a copy color seven times so that the copy color repeats eight times in the palette, the copy color and each duplicated color can hide three bits. The data extraction process is the same as described previously.

To maximize the data hiding capacity, we have to choose appropriate colors as the copy colors. The copy colors are chosen to be the most frequently appearing ones in the cover image. Basically the more times a copy color appears, the more unused squares are allocated to it. But consider the following case: the palette of a cover

image is of the size 256 and there are 3 unused squares. And we also assume that the top three frequently appearing colors are A, B, and C, and the times they appear in the cover image are 200, 150, and 100, respectively. If we allocate all 3 unused squares to A, it means A is used for hiding 2 bits and the hiding capacity will be $2 \times 200 = 400$ bits totally. However, this allocation is not optimal. The maximum data hiding capacity can be achieved when we allocate each color a square such that each color can hide only 1 bit. In such a case the total data hiding capacity becomes $200 \times 1 + 150 \times 1 + 100 \times 1 = 450$ bits, which is larger than the previous one. So we can only grantee that a color with a higher appearing frequency will not be assigned fewer squares than a color with a lower appearing frequency. To find out which assignment is optimal, we use a brute-force method in this study to enumerate all possible frequency-bit combinations. This process seems time-consuming. But actually since the size of the palette is at most 256, the process will not take too long. And since the data embedding process need not be conducted in real time in most applications, the waiting time for using the above-mentioned brute-force method to select the best color copying way is acceptable.

Before the secret messages are really embedded, we have to do a preprocessing to prevent the secret messages from being dismissed by the reordering of the palette. This preprocessing also makes it clear that the bit pattern that a color hides. We use an example to illustrate this. Let A be the copy color with index 10 in the palette and B, C, and D be A's duplicated color with index 11, 12, and 13 in the palette, respectively. The preprocessing is that on scanning the index array, we change the index values of the first four pixels whose color is A. We keep the 1st pixel's index unchanged and modify the 2nd, 3rd, and 4th pixels' to be 11, 12, and 13, respectively. Note this preprocessing does no data hiding. It just means that indices A, B, C, and D are used for hiding bit patterns 00, 01, 10, and 11, respectively. With this preprocessing, if the

palette is reordered such that the indices of A, B, C, and D in the palette become 4, 3, 2, and 1, respectively, we can still know the bit pattern that A, B, C, or D hide. It is due to the fact that we will first meet index 4 on scanning, which means we can extract 00 from every pixel with index 4 on the rest of the scanning. If we do not do the preprocessing, we will not be able to know which bit pattern that a color hides since we will not be able to know which color is the copy color and which are duplicated colors on decoding.

The data extraction process is intuitive. The first thing we have to do is to scan the palette of the stego-image to see how many copy colors are used. Then we construct a decoding table by recording the indices of colors that appear more than once in the palette. During decoding, we scan the stego-image's index array in the raster scan order and look up the decoding table. If the index value being scanned is in the decoding table, several bits will be extracted. The number of bits each time we can extract depends on the appearing times of that copy color in the palette. If the index value being scanned is not in the decoding table, we will just continue to scan the next pixel.

2.2.2 Data Embedding Process

We describe the data embedding process step by step as follows according to the previous discussions.

Algorithm 1. *Hiding secret messages in a GIF cover image.*

Input: A given GIF cover image I and given binary secret messages.

Output: A stego-image S .

Steps:

1. Check the palette of I and count how many squares are used and unused. Record the indices of used squares and unused squares, respectively. Also record the times

each index (pointing to a used square) appears in I .

2. Apply the above-mentioned brute-force method to calculate the maximum data hiding capacity and decide how many bit(s) each copy color can hide.
3. Construct an encoding table with each element of it being the index of a copy color or the index of a duplicated color.
4. Hide the secret message into the image pixels by the principle described in Section 2.2.1 in a raster scan order to obtain S .

2.2.3 Data Extraction Process

We describe the data extraction process step by step as follows.

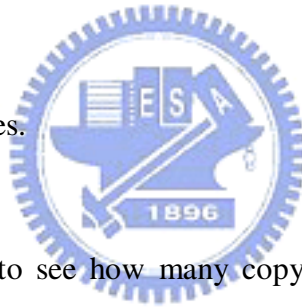
Algorithm 2. *Extracting secret messages from a given stego-image.*

Input: A stego-image S .

Output: Binary secret messages.

Steps:

1. Analyze the palette of S to see how many copy colors exist by examining the repetitions of each color. Then calculate the bits each copy color can hide and record them. Also determine and record the bit pattern that a copy or a duplicated color can be extracted by looking up the first n pixels (which are of the same color as that of copy or duplicated color) if the color repeats n times in the palette.
2. Construct a decoding table and assign each element of it the index of a copy color or a duplicated color.
3. Start to extract secret bits in a raster scan order. If the index happens to be one of the index values in the decoding table, several bits can be extracted according to the record we made in Step 1. If this index is not one of the indices in the decoding table, do nothing but go forward to the next index until the end of the index array.



4. Obtain the binary secret messages.

2.3 Experimental Results

In this section, we show the experimental results on several images in Table 2.1 and the comparison before and after data hiding in

Table 2.2. Note that the file size of the stego-images is slightly larger than that of cover images. This is due to the compression method used on GIFs. Since the method proposed in this study is distortion-free, the experimental results focus on the data hiding capacity.

Table 2.1 Characteristics of tested images and their hiding capacity







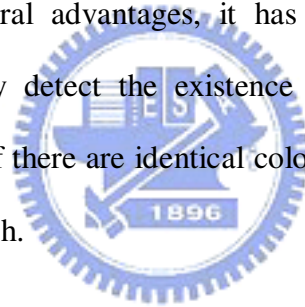
File Name	File Size(KB)	Resolution	Capacity(KB)	Icon
50years	32.8	355x217	31.35	
bbb	2.07	85x28	0.91211	
book	41.6	751x440	157.53	
ipod	71.3	500x532	151.65	
leo	44.9	450x437	105.01	
ship	15.3	300x247	13.819	

Table 2.2 The comparison of cover images and stego-images.

File Name	Cover Image File Size (KB)	Stego-image File Size (KB)	Cover Image	Stego-image
50years	32.8	32.9		
bbb	2.07	2.16		
book	41.6	41.6		
ipod	71.3	71.5		
leo	44.9	44.9		
ship	15.3	15.3		

2.4 Discussion and Summary

In this chapter, a high-capacity, adaptive, and distortion-free data hiding method has been proposed. As shown in the previous sections, some images can even have data hiding capacities larger than the file sizes of themselves. And the proposed adaptive approach yields the maximum data hiding capacity. Since this method establishes a relationship between the palette and the pixel values carrying secret bits, the reordering of the palette does not destroy the secret messages. Although the proposed method is of several advantages, it has a shortcoming on security. A malicious attacker can easily detect the existence of secret messages by simply examining the palette to see if there are identical colors in it. To improve the security is a future work of this research.



Chapter 3

Data Hiding by Stacking Up Two PNG Images

3.1 Introduction

3.1.1 Uses of PNG Images

PNG is the abbreviation of Portable Network Graphics. It is a format of images. As its name suggests, one of its two uses is on the World Wide Web. Since the patent of the LZW (Lempel-Ziv-Welch) lossless data compression algorithm had been enforced by Unisys, the PNG format which uses a non-patented compression method was first developed in early 1995 for replacing the GIF format. Besides the patent-free advantage, PNGs have three other advantages over GIFs: the alpha channel, the gamma correction option, and the two-dimensional interlacing progressive display capability. PNGs that have alpha channels are of variable transparency while GIFs are only of single-level transparency, namely, fully transparent or opaque. As to the progressive display capability, PNGs are superior to GIFs since users can see the complete PNG image earlier than the GIF image when they are loaded at the same time. Details of the display method of PNGs on the Internet will be described in the following section.

The other use of PNG is image editing. PNG is a useful format for the storage of intermediate stages of editing. This benefits from PNG's lossless data compression and its support of up to 48-bit truecolors or 16-bit graylevels. This means that saving, restoring, or re-saving will not degrade its quality, unlike the standard JPEG. Like GIF,

PNG is a raster format, which means that it represents an image as a two-dimensional array of pixels. PNG is not explicitly a vector format which allows users to store shapes (lines, boxes, etc.) and can be scaled arbitrarily without any loss of quality.

3.1.2 Properties of PNG Images

The fundamental building block of PNG images is 'chunk.' With the exception of the first 8 bytes in the file, a PNG image consists of nothing but chunks. Every chunk has the same structure: a 4-byte length, a 4-byte chunk type, a 4-byte cyclic redundancy check value (CRC). The simplest PNG file contains three chunks. They are the image header chunk, IHDR; the image data chunk, IDAT; the end-of-image chunk, IEND. An IHDR must be the first chunk in a PNG image, and it includes all of the details about the type of the image: its height and width, pixel depth, compression and filtering methods, interlacing method, whether it has an alpha (transparency) channel, and whether it is a truecolor, grayscale, or palette image. An IDAT contains all of the image's compressed pixel data. Each IDAT can contain at most 2 gigabytes of compressed data. An IEND is the simplest chunk of all. It contains no data, and just indicates that no chunk follows. These three chunk types are sufficient to build truecolor and grayscale PNG files, with or without an alpha channel, but palette-based images require one more type of chunk: PLTE, the palette chunk, which consists of lots of squares like GIF. But the number of squares is not necessarily a power of two.

A byte called ColorType in the IHDR chunk indicates what kind the image is of. If ColorType equals 0, the image type is gray-level; if ColorType equals 2, the image type is truecolor; if ColorType equals 3, the image type is palette; if ColorType equals 4, the image is gray-level with an alpha channel; if ColorType equals 6, the image is truecolor with an alpha channel. As to the pixel depth, it is indicated by another byte named Bit Depth. A palette PNG image can be of pixel depth 1, 2, 4 or 8; a gray-level

one can be of pixel depth 1, 2, 4, 8 or 16; and a truecolor one can be of 8 or 16.

Another byte in the IHDR chunk, called Interlace, indicates how the PNG image is displayed. If the byte is 0, the image is displayed with interlacing; if it is 1, the image is displayed without interlacing. The interlacing method of PNG is a two-dimensional scheme with seven passes, known as the Adam7 method (after its inventor, Adam Costello). The primary benefit of the PNG's two-dimensional interlacing, contrasting with the GIF's one-dimensional scheme, is that one can view a crude approximation of the entire image roughly eight times as fast.

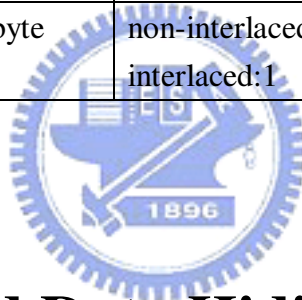
The PNG specification defines a single compression method, the 'deflate' algorithm, for all image types. Although not the best compression algorithm known so far, 'deflate' has a very desirable mixture of characteristics: high reliability, good compression, good encoding speed, excellent decoding speed, and minimal overhead on incompressible data. The IHDR chunk is illustrated in

Table 3.1.

The above three types of chunks, IHDR, IDAT, and IEND, are named *critical chunks*. There are 10 other types of chunks that do not necessarily exist in a PNG image. They are bKGD (background color), cHRM (primary chromaticities and white point), gAMA (gamma), hIST (image histogram), pHYS (physical pixel dimensions), sBIT (significant bits), tEXt (textual data), tIME (image last-modification time), tRNS (transparency), and zTXt (compressed textual data), respectively. They are named *ancillary chunks*.

Table 3.1 The structure of IHDR chunk.

Field Name	Byte Number	Comment
Width	4 bytes	width of the image, measured in pixels.
Height	4 bytes	height of the image, measured in pixels.
Bit depth	1 byte	palette image:1,2,4,or 8 gray-level image:1,2,4,or 8 truecolor image:8 or 16
ColorType	1 byte	gray-level image:0 truecolor image:2 palette image:3 gray-level image with alpha channel:4 truecolor image with alpha channel:6
Compression method	1 byte	LZ77-derived algorithm, <i>deflate</i> .
Filter method	1 byte	method of filtering
Interlace method	1 byte	non-interlaced:0 interlaced:1



3.2 Proposed Data Hiding Method

3.2.1 Proposed Idea

In this study, we pile up two gray-level PNGs for data hiding and the upper image is created with an alpha channel. The upper one is called a *foreground image* while the lower one is called a *background image*. We name the resulting new image a *piled-up image*. According to the PNG standard, if the upper image contains an alpha channel or is with some pixels fully transparent or opaque, the color that a transparent pixel of the piled-up image presents is calculated in the following way.

First, denote F and A , respectively, as the intensity and the alpha value that a

pixel of the foreground image presents; B as the intensity that a pixel of the background image presents; and P as the intensity that a pixel of the piled-up image presents. Then the grayscale value P of a pixel in the piled-up image with transparency is computed by

$$P = (F \times A + B \times (255 - A)) / 255.$$

This formula is adopted by most graphic cards for showing the pixel value on the screen. The alpha value A can be treated as the weight that determines how many percentages the foreground and background colors should present. Note that the foreground image and the background image need not be of the same size. But at least the background image should be no smaller than the foreground image.

Our idea of data hiding is to embed secret messages in the alpha channel of the foreground image with the 5-bit LSB method. For each pixel, since bit embedding in the alpha channel might result in an increase or decrease of the alpha value, the value P will also be increased or decreased. In order to increase the quality after data embedding, we make use of the above-mentioned formula to increase the PSNR. The details are described as follows.

For each pixel, let the alpha value after embedding be A'. We adjust F and B to be F' and B', respectively, according to the difference D defined by $F \times A + B \times (255 - A)$ minus $F \times A' + B \times (255 - A')$. If D is greater than zero, we first subtract $\text{floor}(D/A')$ from F to form F'. Then we define the difference D' by $F \times A + B \times (255 - A)$ minus $F' \times A + B \times (255 - A)$. Since D' must be greater than zero, we further subtract $\text{floor}(D'/(255-A'))$ from B to form B'; If D is smaller than zero, we first add $\text{floor}(D/A')$ to F to form F' and since D' must be smaller than or equal to zero, we further add $\text{floor}(D'/(255 - A'))$ to B to form B'.

The philosophy of the above idea is as follows. We first reduce the difference

between $F \times A + B \times (255 - A)$ and $F \times A' + B \times (255 - A')$ by adjusting the intensity F to be F' . Since F' is gained from F plus or minus $\text{floor}(D/A')$, we can guarantee that D' and D are of the same sign. Keeping the consistency of the sign makes it easier to program. More specifically, if F' is produced by an adding operation on F , B' will also be produced by an adding operation on B ; if F' is produced by a subtracting operation on F , B' will be produced by a subtracting operation on B .

It is common that two or more images are piled up on web pages. Usually a web page consists of a background image and some foreground images. If we want to hide secret messages in pixels that are formed by piling up two images on a web page, we can use the method mentioned above.

3.2.2 Data Embedding Process

The data embedding process is described step by step in Algorithm 1 as follows. Secret messages are embedded in a raster scan order.

Algorithm 3.1. *Hiding secret binary messages in two images.*

Input: A cover image that is formed by piling up a foreground image with an alpha channel and a background image.

Output: A stego-image S made of two piled-up images.

Step:

1. Use a key to randomize the secret data to be embedded and then embed five randomized secret bits in the alpha channel of the foreground image of the current pixel.
2. Handle the “salt-and-pepper” problem for the alpha channel with the following rule: Add 32 to those values that are smaller than 0. Subtract 32 from those values that are larger than 255.

- For each pixel, do the following operations:

compute $D = (F \cdot A + B \cdot (255 - A) - F' \cdot A' + B' \cdot (255 - A'))$;

if $(D > 0)$, compute

$$F' = F - \text{floor}(D/A');$$

$$D' = (F \cdot A + B \cdot (255 - A) - F' \cdot A' + B' \cdot (255 - A));$$

$$B' = B - \text{floor}(D'/(255 - A'));$$

else if $(D < 0)$, compute

$$F' = F + \text{floor}(D'/A');$$

$$D' = (F \cdot A + B \cdot (255 - A) - F' \cdot A' + B' \cdot (255 - A));$$

$$B' = B - \text{floor}(D'/(255 - A')).$$

- Obtain the stego-image S .

3.2.3 Data Extraction Process

We describe the data extraction process as follows.

Algorithm 2. *Extracting secret from a given stego-image.*

Input: A piled-up stego-image S .

Output: Binary secret messages.

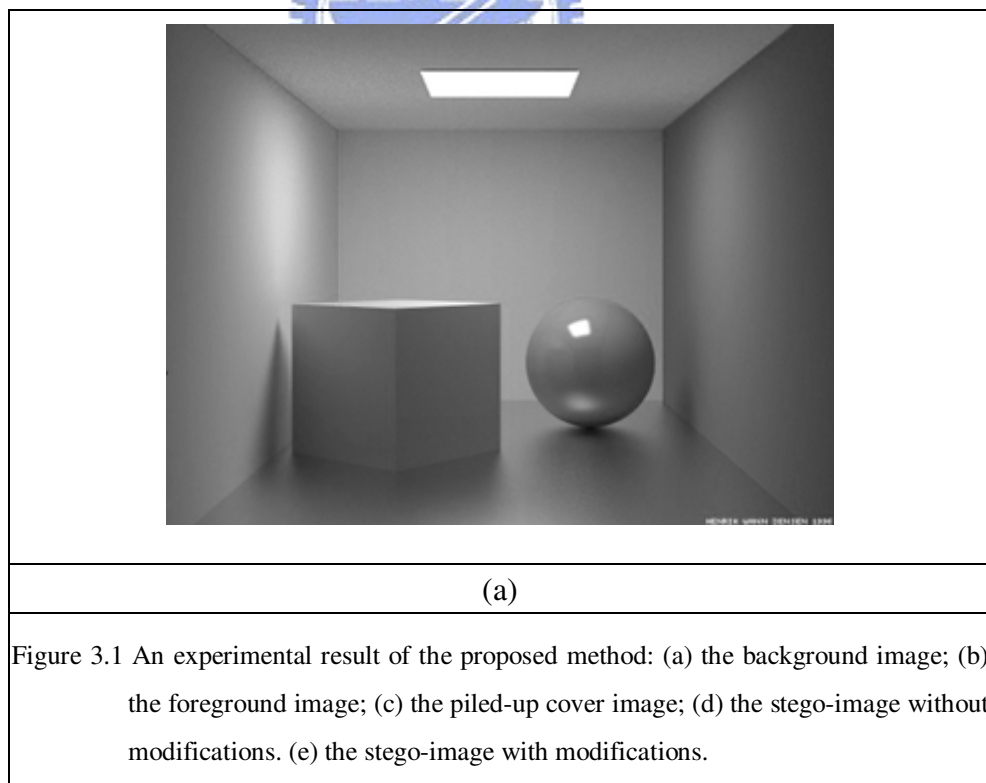
Steps:

- Extract five bits from the alpha channel of the current pixel by A' modulo 32.
- Combine the secret bits together extracted from each pixel to form the randomized secret messages.
- Use the same key used in the data embedding process to get the real secret message.

3.3 Experimental Results

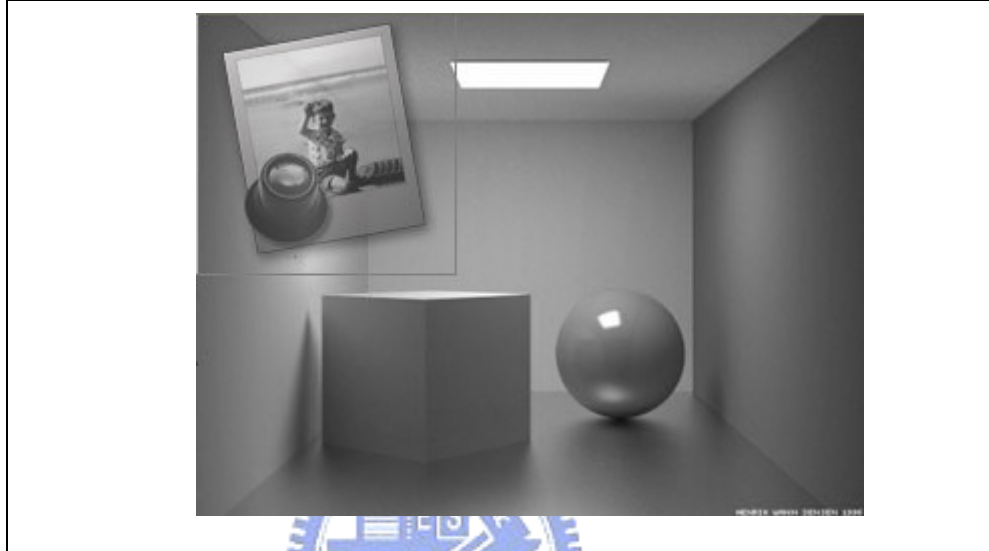
In this section, we show the experimental results on some background and foreground images. We show the total hiding capacity that each stego-image can hide and also show the PSNR before and after adjusting the foreground and background values.

An experimental result is shown in ~~錯誤! 找不到參照來源~~. Figure 3.1(a) shows a 333x250 background cover image, box.png. Figure 3.1(b) shows a 128x128 foreground image, photo.png. And Figure 3.1(c) shows the piled-up image where no secret bit is embedded. Figure 3.1(d) shows the stego-image without adjusting the foreground and background intensities. Figure 3.1(e) shows the adjusted stego-image. The data hiding capacity is $128 \times 128 \times 5 = 10\text{KB}$. The PSNR is 30.8941 without adjusting the foreground and background intensities and 47.6422 with adjusting. We pile the foreground image on the background image at the upper leftmost corner.





(b)

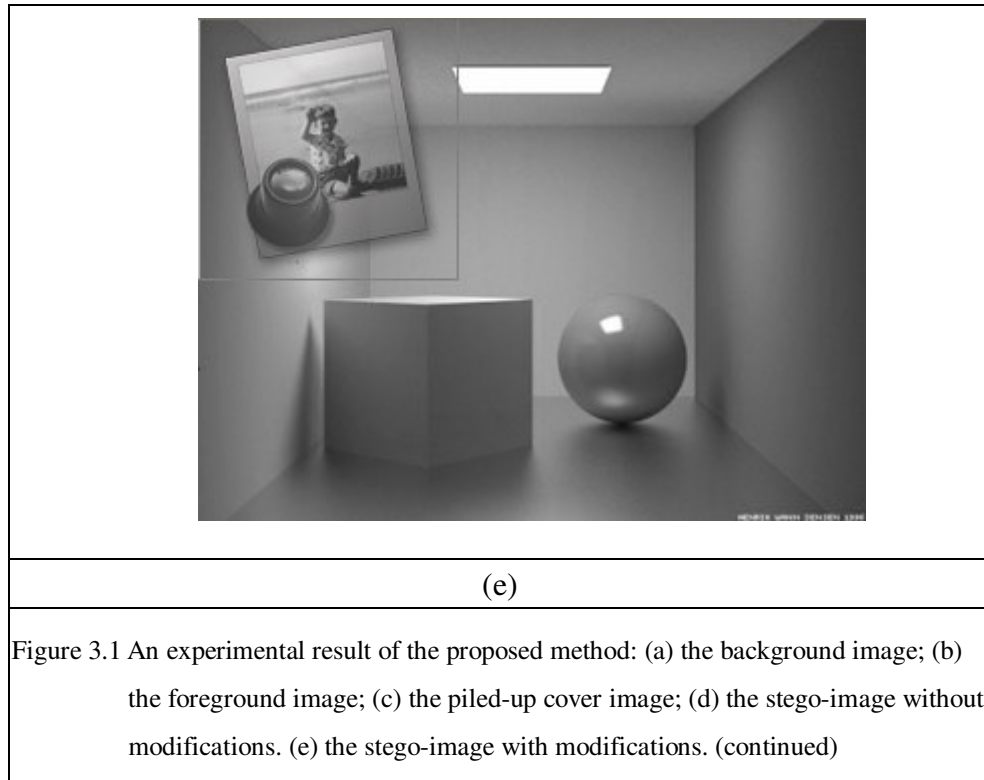


(c)



(d)

Figure 3.1 An experimental result of the proposed method: (a) the background image; (b) the foreground image; (c) the piled-up cover image; (d) the stego-image without modifications. (e) the stego-image with modifications. (continued)

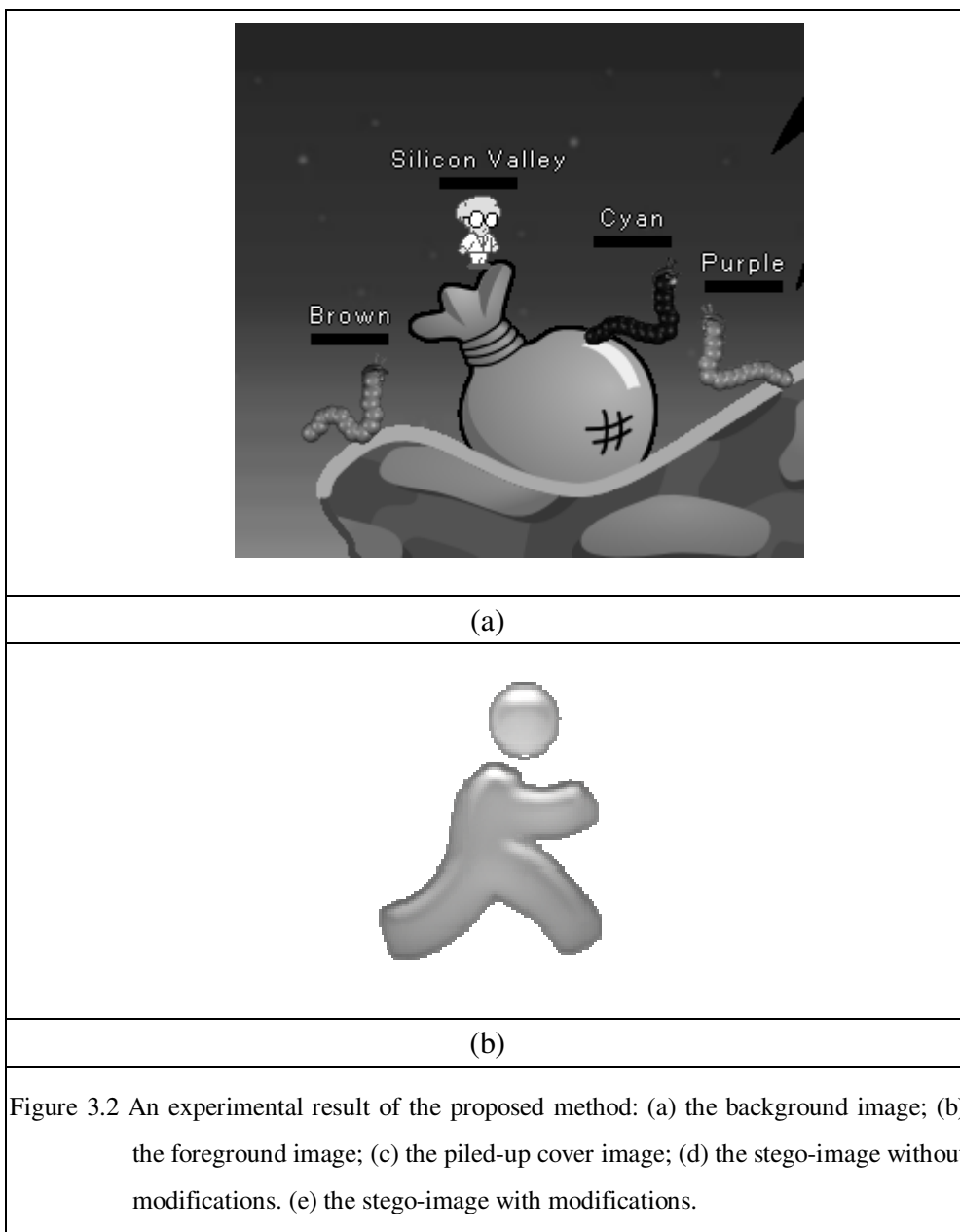


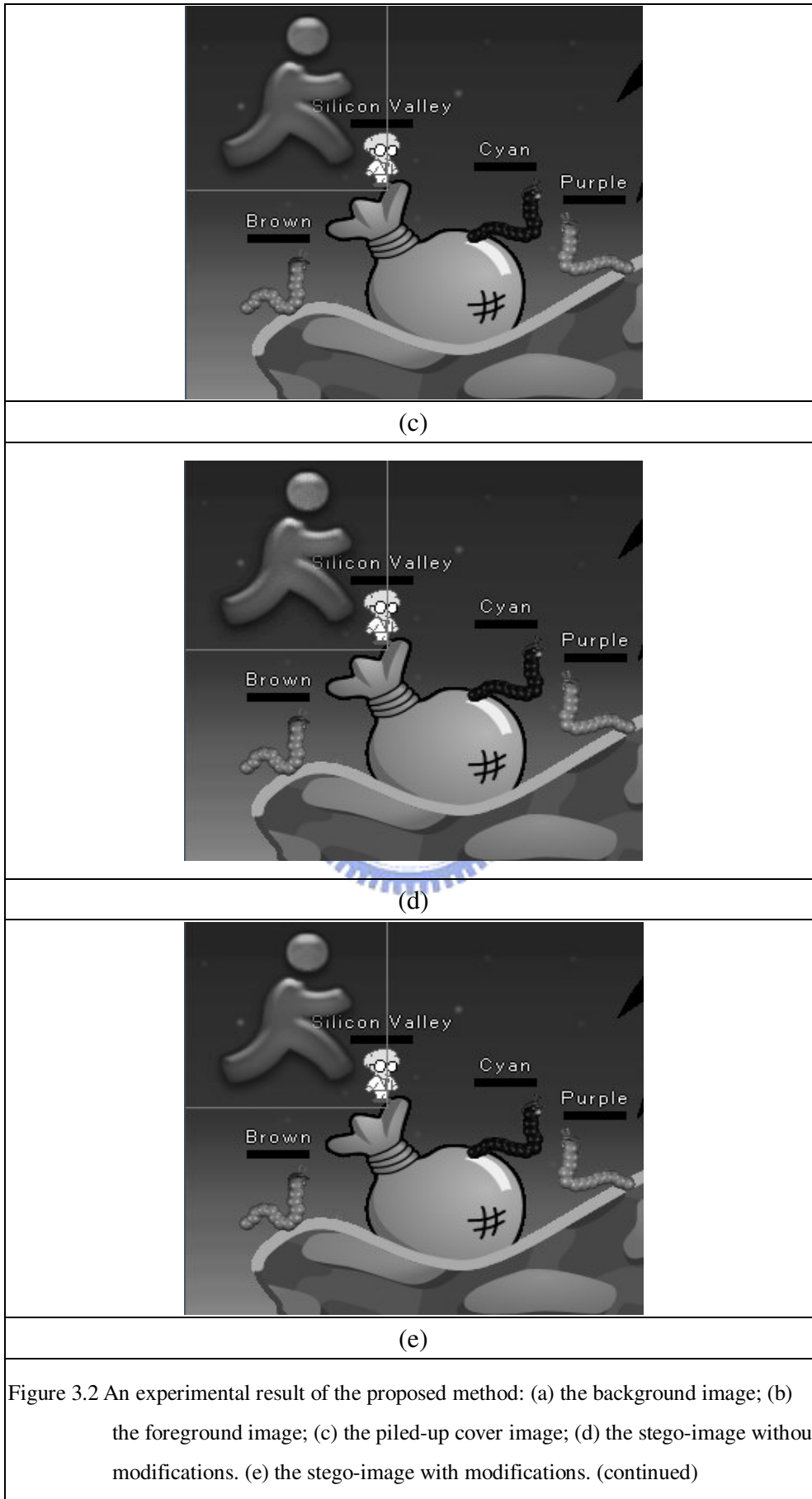
Another experimental result is shown in Figure 3.2. Figure 3.2(a) shows a 584x457 background cover image, pirate.png. 錯誤! 找不到參照來源。 (b) shows a 128x128 foreground image, walker.png. And Figure 3.2(c) shows the piled-up image where no secret bit is embedded. Figure 3.2(d) shows the stego-image without adjusting the foreground and background intensities. Figure 3.2(e) shows the adjusted stego-image. The data hiding capacity is $128 \times 128 \times 5 = 10\text{KB}$. The PSNR is 37.8782 without adjusting the foreground and background intensities and 59.4150 with adjusting. We pile the foreground image on the background image at the upper leftmost corner.

3.4 Discussion and Summary

In this chapter, a method that embeds secret data in piled-up images has been

proposed. This method can be used in data hiding on web pages. In this method, only the alpha channel of a given image is used for hiding data. Note that we adjust the intensity of the foreground image and the intensity of the background image without hiding secret data in them. We can make use of both of the intensities to increase the data hiding capacity. But this may result in a decreasing of the PSNR value. To develop a method that not only has a high hiding capacity but also a high PSNR value after embedding is a good topic for future works of after this study.





Chapter 4

Fast Search Methods via Hidden Messages in Images

4.1 Introduction

4.1.1 Need of Fast Search of Desired Images

Digital image technology continues progressing and image libraries and databases are widely used in recent years. Images are created or available much more easily than they are in the past ten years. One can create images in the following two main ways. One is to create images by a scanner to scan paper and store what is scanned in a digital form. The other is to create images by digital cameras. A digital camera usually stores what has been shot in a compressed form to save spaces. It is common that hundreds of pictures are created during one's journey. Plenty of images are also easily available through the Internet. A part of the functionalities of Google is one of the examples. People can get images they want by using keywords to search on Google at the IP address <http://images.google.com.tw/>. So it is critical to manage images more efficiently and effectively. Sometimes people want to load and view a desired image from their disks but it is quite hard or even impossible for them to remember all of the names of the images or to check all of the images in their computer. To develop a fast method for searching desired images is the goal of this study.

4.1.2 Possible Techniques for Speeding Up Searches

Databases are usually used for storing huge amounts of data or records and a program can consult it to answer queries. The records retrieved in answer to queries become information that can be used to make decisions. The computer used to manage and query a database is known as a database management system (DBMS). One of the possible techniques for speeding up searches is to use an image database to record and manage images in it.

4.2 Proposed Search Method

4.2.1 Proposed Ideas

In this study, we propose two methods that can achieve faster searches for image databases. The idea is described as follows.

Before an image is stored into the image database, users have to enter some comment and we use data hiding techniques to hide the comment into that image for the search purpose. That means what we really store is a *stego-image* instead of a *cover image*. This scheme also has the advantage of requiring no additional space to save the comment for each image. In this study, we use BMPs as cover images and stego-images. And we conduct the 4-bit least-significant-bit (LSB) method to hide a comment into the red channel of a BMP file. Then we propose two methods to accomplish fast searches of the image database.

A. Method 1 --- non-block-level search

The first method we propose to conduct fast searches is called “the non-block-level search method.” We achieve this by storing additional information

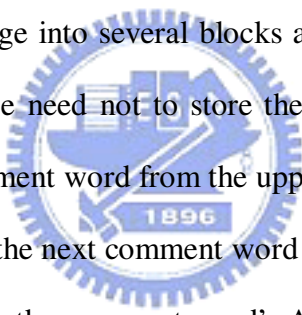
into images. We store two additional pieces of data so that we can search for desired images in a two-level hierarchical way. The first-level hierarchy is that, for each single word of the comment, we sum up the ASCII code values of all the letters of that word and store the value of the sum modulus 255. Actually, this is a hashing approach, and what we actually do is to hash the ASCII sum into a hash table of size 255. The second hierarchy is that, for each single word of a comment, we calculate the word length of it (i.e., the number of letters that make up the whole word) and store the result. So what we finally store are the ASCII sum, the word length, and the word itself. This is shown in Figure 4.1. In this study, we only allow users to search for desired images by a single key word and we return the file names of desired images as output as long as a word of their comment matches the key word. After the single key word is entered, we calculate the ASCII sum and the word length of it and then start the search. First, we compare the ASCII value of the current comment word and that of the key word. If the two values match, we further compare the word lengths of them and if the two values still match, we then compare the two word letter-by-letter. And of course, if all of the letters and the positions of the letters of the two words are the same, the user will get the desired images and their titles. If any of the ASCII sum value, word length, and the letters being compared of the two words does not match, we just go further to compare the next comment word with the key word until a comment word that matches or until the end of the comment of the image.

In the above process, since we record the word length, we do know how many letters we should skip if a mismatch occurs. To store the word length is the key to speeding up searches since it erases the comparison times of letters to facilitate looking forward to the next comment word. This is by far an advantage when the comment word is of a long length. Consider the following case: suppose that a part of the comment is “misunderstandings with...” and the key word is “my.” We first store

the comment, say, in the form of “57 16 misunderstandings 189 4 with...” and transform the key word, say, to the form of “230 2 my”. Since 57 is not equal to 230, we know that a mismatch occurs and we can directly skip at least 16 letters and start to compare 57 and 184. Without the store of the word length, we will have to know the starting point of the next comment word by checking each letter until the appearance of the character, *space*.

B. Method 2 --- block-level search

The first method is a sequential approach, which means not until the current comment word is compared can we process the next comment word. The second method is a block-level search method which can be applied in a *non-sequential order*. First, we divide the cover image into several blocks and each block is used to hide a single comment word. And we need not to store the length of comment words this time; since we store each comment word from the upper left most pixel of a block, we know exactly the place where the next comment word will appear. The only additional information we have to store is the comment word’s ASCII sum modulus 255. This is shown in Figure 4.2. The overall comparison procedure is the same as the first method, so we do not describe it in detail.



ASCII sum	Word length	Comment word
-----------	-------------	--------------

Figure 4.1 The data structure of the non-block-level method.

ASCII sum	Comment word
-----------	--------------

Figure 4.2 The data structure of the block-level method.

4.2.2 Non-block-Level Search Method

In this section, we describe the algorithms for implementing the non-block-level search method as follows.

Algorithm 4.1. *Hiding comment words in a BMP cover image file for later search.*

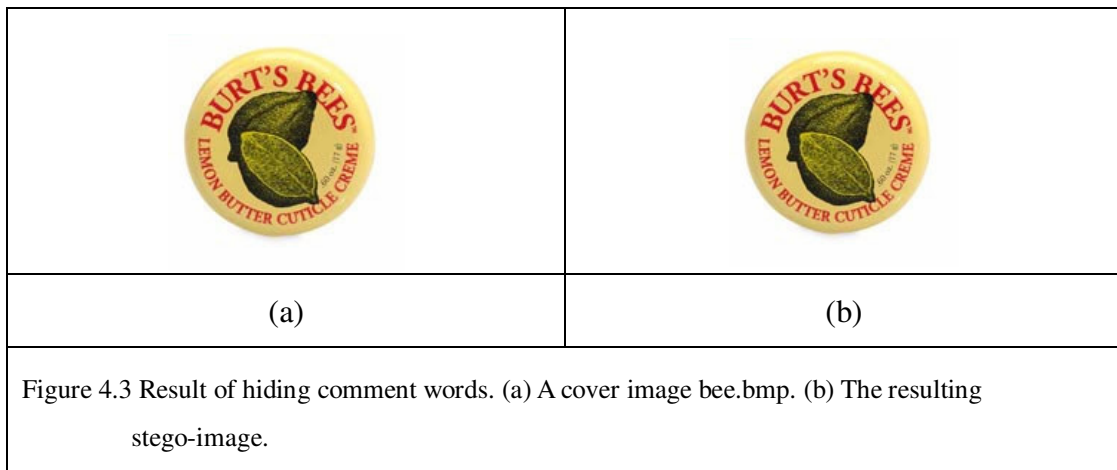
Input: A given BMP cover image and comment words to be embedded.

Output: A stego-image S .

Steps:

1. Grab a comment word and calculate both the ASCII sum modulus 255 and the word length of it.
2. Embed the ASCII sum modulus 255, the word length, and the comment word itself in order by applying the 4-bit LSB method to the red channel of the cover image.
3. Repeat the previous two steps until all the comment words are embedded.
4. Take the resulting image as the desired stego-image S .

As an illustration of the data embedding result of applying the above algorithm, Figure 4.3(a) shows a cover image and Figure 4.3(b) shows the resulting stego-image after embedding the comment “This is a lemon butter cuticle crème.”



Algorithm 4.2. *Searching for desired images.*

Input: A key word used for search and a set of stego-images forming an image database.

Output: Desired images with comments matching the key word.

Steps:

1. Calculate the ASCII sum modulus 255 and the word length of the key word.
2. Grab an image from the image database.
3. Compare the key word and the comment word in the following order:
 - (1) compare the ASCII sums modulus 255;
 - (2) compare the word lengths;
 - (3) compare the words letter by letter.

As long as the values or the letter being compared fails to match, skip to check the next comment word. Return the file name of the image if all of the comparisons succeed. Grab the next image if all of the comment words of the current image fail to match the key word.

4. Repeat Step 1, 2, and 3 until every image in the database has been checked.
5. If no desired image exists, return the message of 'no such file.'

4.2.3 Block-Level Search Method

In this section, we describe the algorithms for implementing the block-level search method as follows.

Algorithm 4.3. *Hiding comment words in a BMP cover image file for later search.*

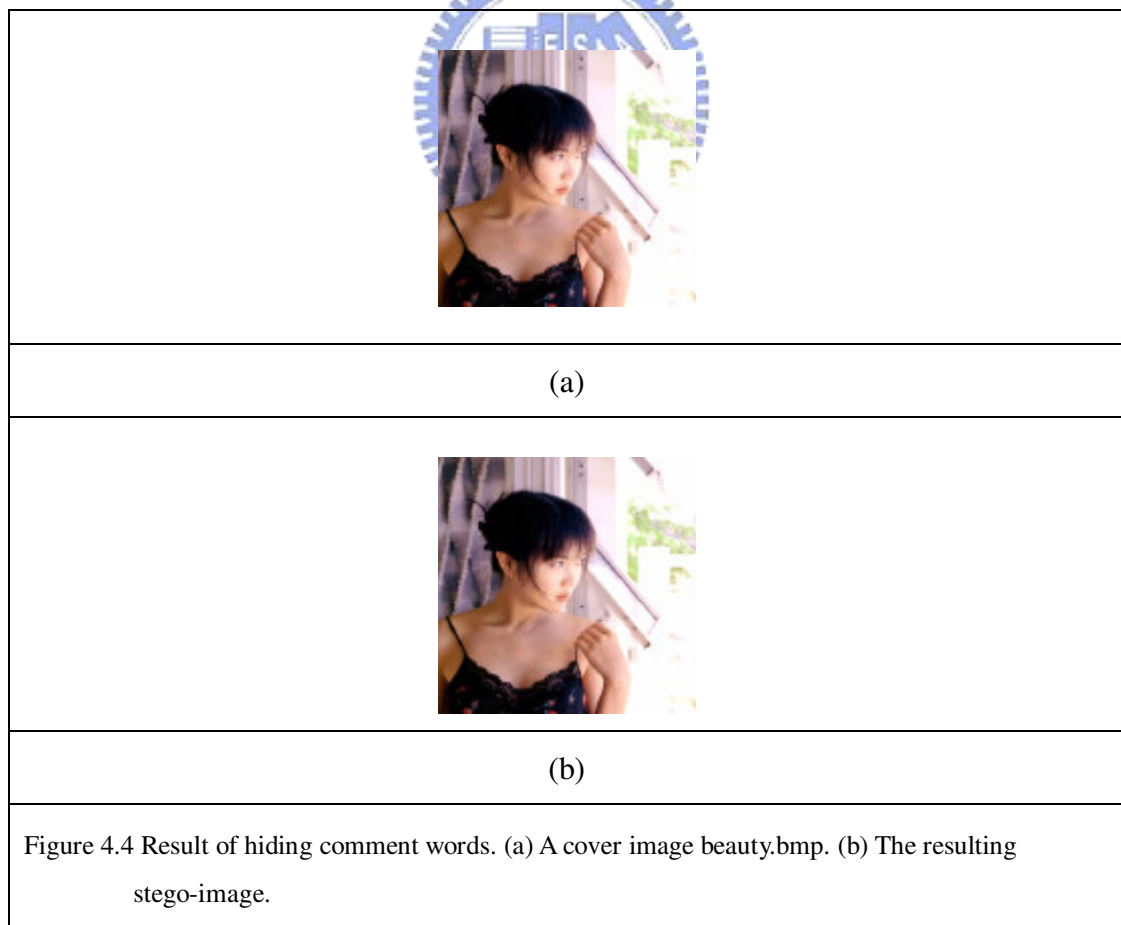
Input: A given BMP cover image and some comment words to be embedded.

Output: A stego-image S .

Steps:

1. Divide the image into several blocks and each block is of size $n \times n$ or $m \times n$, where m is not equal to n .
2. Grab a comment word and calculate the ASCII sum modulus 255 of it.
3. Embed the ASCII sum modulus 255, and the comment word itself in the current block, starting from the upper left most pixel of the block, by applying the 4-bit LSB method to the red channel of the pixel's color values.
4. Repeat the previous two steps until all the comment words are embedded.
5. Take the resulting image as the desired stego-image S .

As an illustration of the data embedding result of applying the above algorithm, Figure 4.4(a) shows a cover image and Figure 4.4(b) shows the resulting stego-image after embedding the comment "This is a Japanese girl beside the window."



Algorithm 4.4. *Searching an image database for desired images.*

Input: A key word used for search and a set of stego-images forming an image database.

Output: Desired images with their comments matching the key word.

Steps:

1. Calculate the ASCII sum modulus 255 of the key word.
2. Grab an image from the image database.
3. Compare the key word and the comment word with the following order:
 - (1) compare the ASCII sums modulus 255 after extracting the value of the ASCII sums modulus 255 from the 4 LSBs of the pixels of the image blocks;
 - (2) compare the words letter by letter after extracting the current comment word from the 4 LSBs of the pixels of the image blocks.As long as the values or the letters compared fails to match, skip to check the next comment word. Return the file name of the image if all of the comparisons of a comment word match. Grab the next image if all of the comment words of the current image fail to match the key word.
4. Repeat Steps 1, 2, and 3 until every image has been checked.
5. If no desired image exists, return the message of ‘no such file.’

4.3 Experimental Results

In this section, we show some experimental results. We show a result of non-block-level method first and then that of the block-level one. For the first experimental result of the non-block-level method, we embed different comments into a particular image. The smaller the difference is, the fewer times we need to judge

whether this image is desired or not. The particular image we use is Lena.bmp (shown in Figure 4.5) with its size 512×512 and the first comment we embed is “Lena is a model.” And the second comment we embed is “The girl who lives in the U.S. and studies computer science is called Lena Wang.” Then we use “Lena” as the key to search. Note that the first comment has “Lena” as the first word while the second comment has “Lena” as the second last word. The time elapsed using the first comment to search for the desired image is 0.042153 seconds while the time elapsed using the second comment to find out the desired image is 0.061118 seconds, which is less than a 0.1 second.



Figure 4.5 The particular image, Lena.bmp.

For the second experimental result of the block-level method, we tried to find two desired images (Lena1.bmp and Lena2.bmp) from an image database containing 200 images. Some of the images in the database are shown in Figure 4.6(a). The key word is the same as the previous one (i.e., the key word is Lena) and the comments we embed in the two images are again “Lena is a model.” and “The girl who lives in the U.S. and studies computer science is called Lena Wang.”, respectively. The times elapsed to find out the two desired images are around 0.04842 and 0.1046 seconds, respectively. Figure 4.6(b) shows the search results, including the two images with the

key word ‘Lena.’

			
Lena1.bmp	Lena2.bmp	AMIGO001.bmp	AMIGO002.bmp
			
AMIGO003.bmp	AMIGO004.bmp	AMIGO005.bmp	AMIGO006.bmp
			
AMIGO007.bmp	AMIGO008.bmp	AMIGO009.bmp	AMIGO0010.bmp
			
AMIGO0011.bmp	AMIGO0012.bmp	AMIGO0013.bmp	AMIGO0014.bmp
(a)			

Figure 4.6 Image database used in the experiment. (a) Some images in the database and the first two images are Lena1.bmp and Lena2.bmp with comments “Lena is a model” and “The girl who lives in the U.S. and studies computer science is called Lena Wang.”, respectively. (b) Two desired images found in 0.04842 and 0.1046 seconds, respectively.



	
Lena1.bmp	Lena2.bmp
(b)	

Figure 4.6 Image database used in the experiment. (a) Some images in the database and the first two images are Lena1.bmp and Lena2.bmp with comments “Lena is a model” and “The girl who lives in the U.S. and studies computer science is called Lena Wang.”, respectively. (b) Two desired images found in 0.04842 and 0.1046 seconds, respectively. (continued)

4.4 Discussion and Summary

In this chapter, two methods for fast search of image databases have been proposed. We hide comments inside images so that we do not need additional space to save the comments and as shown in the previous section. Although we can find out desired images in a short time by using either one of the methods, we still feel that there is space to improve the efficiency of both methods. To make the search faster for both methods is our future work.

Chapter 5

Conclusions and Suggestions for Future Works

5.1 Conclusions

In this study, we have proposed several methods for different purposes, including a data hiding method for GIFs, a data hiding method for use on web pages, and two methods for fast image searches. All of them are implemented in a PC environment.

The proposed data hiding method for GIFs can be used to hide huge amounts of data in a GIF file without distorting the image. For the data hiding method used on web pages, we pile up two gray-level PNG images and hide secret data in the alpha channel of the foreground image. And then we adjust both the intensity of the foreground image and that of the background image such that the intensity that the pixel presents after embedding will not be too far away from the original intensity. In this way, we can hide secret messages in a web page without being detected easily.

As to the two methods for fast searches of images in databases, one method is a non-block-level method while the other is a block-level method. Both methods use the comments hidden in images as indices to search and both can find out desired images in a very short time.

5.2 Suggestions for future works

Several suggestions for future researches are listed as follows.

1. Developing a distortion-free and high-capacity data hiding method for GIF images, such that the embedded secret messages are not easily detected by an attacker.
2. Developing high-capacity data hiding methods on other image formats.
3. Developing a data hiding method, based on the proposed concept of piling up two PNGs, which hides secret messages in the alpha channel and the intensity channel of the foreground image, as well as the intensity channel of the background image without producing serious artifacts in the stego-image.
4. Developing faster search methods for the BMP image file and other image formats.



References

- [1] E. H. Adelson, "Digital signal encoding and decoding apparatus," *U.S. Patent 4939515*, 1990.
- [2] D. C. Lou and J. L. Liu, "Steganographic Method for Secure Communications," *Computers and Security*, Vol. 21, Issue 5, pp. 449-460, October 1, 2002.
- [3] Y. K. Lee and L. H. Chen, "High Capacity Image Steganographic Model," *Vision, Image and Signal Processing, IEE Proceedings-*, Vol. 147, No. 3, pp. 288-294, June 2000.
- [4] N. F. Johnson and S. Jajodia, "Steganalysis: The Investigation of Hidden Information," *Information Technology Conference*, 1998. IEEE, pp.113-116, 1-3 Sep 1998.
- [5] M. D. Swanson, B. Zhu, and A. H. Tewfik, "Robust Data Hiding for Images," *IEEE Digital Signal Processing Workshop Proceedings*, pp.37-40, 1-4 Sep 1996.
- [6] Y. J. Cheng, C. Y. Yin, D. C. Wu, and W. H. Tsai, "Copyright Protection and Authentication of Image Information in Digital Libraries," *Proceedings of 2001 Conference on Trends and Perspectives in Digital libraries and Digital Museums*, Hsinchu, Taiwan, Republic of China, pp. VIII 1~11 (in Chinese).
- [7] Chen and Lee, "Non-Photorealistic 3D Rendering in Chinese Painting Style," *M. S. Thesis*, Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan, Taiwan, Republic of China, July 2003.
- [8] T. K. Yen, "Image Hiding by Random Bit Replacement and Frequency Transformations," *M. S. Thesis*, Department of Computer and Information Science, National Chiao Tung University, Taiwan, Republic of China, 1998.

- [9] N. Abdulaziz and K. K. Pang, "Wavelet Transform and Channel Coding for Data Hiding in Video," *Info-tech and Info-net, 2001. Proceedings. ICII 2001 - Beijing. 2001 International Conferences*, Vol. 2, 2001, pp. 791–796.
- [10] H. Y. Chang, "Data Hiding and Watermarking in Color Images by Wavelet Transforms," *M. S. Thesis*, Department of Computer and Information Science, National Chiao Tung University, Taiwan, Republic of China, 1999.
- [11] G. Xuan, J. Zhu, J. Chen, Y.Q. Shi, Z. Ni, and W. Su, "Distortionless Data Hiding Based on Integer Wavelet Transform," *Electronics Letters*, Department of Computer Science, Tongji University, Shanghai, China, 2002.
- [12] M. Kwan, "GIF Colormap Steganography," 1998.
- [13] R. Machado, "EzStego, Stego Online, Stego", 1997.
- [14] H. Repp, "Hide4PGP," 1996.
- [15] A. Brown, "S-Tools for Windows," 1996.
- [16] C. Maroney, "Hide and Seek," 1994-1997.
- [17] N. F. Johnson and S. Jajodia, "Steganalysis of Images Created Using Current Steganography Software," *New York: Springer-Verlag*, Vol. 1525, Lecture Notes in Computer Science, pp. 273–289, 1998
- [18] J. Fridrich and R. Du, "Secure Steganographic Methods for Palette Images," *Proc. 3rd Int. Workshop Information Hiding*, Dresden, Germany, pp. 47–60, 1999.
- [19] L. Hongmei, Z. Zhefeng, H. Jiwu, H. Xialing, and Y. Q. Shi, "A High Capacity Distortion-free Data Hiding Algorithm for Palette Image," *Proc. IEEE Int. Symp. on Circuits and Systems 2*, pp. 916–919, 2003.