Project ImagIn

Multimedia Communications, ETH Zürich

Martin Lanter

lanterm@student.ethz.ch

January 4, 2013

1 Introduction

Images are part of our daily life. We see them every day and believe that they contain not less and not more than our eyes percept. This is not true. ImagIn is a software written in Scala that lets you put more into an image than what it shows. ImagIn encodes data within an image while preserving the most significant part of it. The resulting image can be sent to another person who then can separate the date again using ImagIn. Hiding data inside an image is called Steganography and is a form of security through obscurity.

2 Implementation

One pixel of an image is built of 3 bytes where each byte determines the color value of red, green or blue respectively. A byte consists of 8 differently significant bits. Let us assume we have a byte that represents a color value, e.g., 0x10000010 (= 130). If we change the first 1 of this byte the color turns almost black. If we, however, change the second 1, the value does almost not change. Using the least significant bits to encode data changes the image as little as possible. A user can choose 1 to 7 bits per byte to encode data. Given an RGB image of size $w * h$ and using b bits we can encode $w * h * 3 * b/8$ bytes. Inside an adequate 400*300 pixel image and 4 hiding bits we are able to indiscernibly encode a payload of 180 KB.

The encoding within the image consists of a small header and the payload. The header consists of the encoded number of hiding bits and the encoded size of the payload. Given a $w*h$ pixel RGB image and b hiding bits, we use the first pixel to encode the number of hiding bits. Since b is at least 1, the last bit of every byte certainly can be used for encoding. Having one pixel, i.e. three bytes, is exactly sufficient to encode the number of hiding bits (1 to 7). Second, we encode the payload size. Currently, ImagIn uses 3 bytes, i.e. 24 bits, to encode the payload size. Therefore the maximum payload size is 16 MB but could easily be extended. When using 7 hiding bits, we require 2 pixels, i.e. 6 bytes of which 4 are used. When using 1 hiding bit, we require 8 pixels (24 bytes).

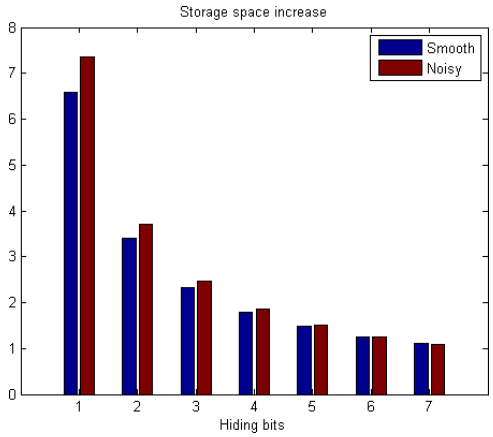
The remainder of the image is used for the payload. If the payload is smaller than the available space, the unused encoding bits are left blank. One might of course change that and insert for instance the bits from the original image or random bits. Finally, ImageIn uses the lossless PNG compression to store the new image with the encoded data to disk.

3 Evaluation

I use two images to evaluate ImagIn: An image of the ETH main building with a smooth sky and a noisy image of a forest. I encode the same bytes in both using 1 to 7 bits to hide the data. Figure 2 shows the two images with encoded bytes using different number of hiding bits. From my own subjective point of view I would claim that using 1 or 2 bits of the smooth image does not change the image recognizably. When using 3 bits one can see some artifacts in the blue smooth sky and when using 4 bits the sky visibly decomposes into pieces. With even more hiding bits, the image clearly loses quality. In the noisy forest image, 4 hiding bits are still almost not recognizable. When using 5 bits, there are some artifacts on the tree on the right but only when using 6 or 7 bits the image literately falls to bits.

Hide bits	Smooth	Noisy
1	0.000008	0.000008
2	0.000039	0.000039
3	0.00016	0.00016
4	0.00064	0.00066
5	0.0025	0.0027
6	0.0093	0.011
7	0.040	0.037

a) Mean squared error



b) Storage space increase

Figure 1: a) The mean squared error of the images compared to the original. The MSE does not correlate well with the image quality a human observes. b) The increase in storage size required to hide the date within an image.

As a more objective measure of quality loss I have computed the mean squared error (MSE) of each encoded image compared to the original. Table 1 a) shows that the MSE strongly depends on the number of hiding bits. It is interesting that it is not always the same image with the higher MSE. For 7 hiding bits, the MSE of the noisy forest is smaller but for 6 hiding bits, the MSE of the smooth image is. However, the MSEs for both images are very similar while my subjective quality estimation is not at all. Therefore, the MSE does not correlate very well with the actual quality loss a human observes.

Figure 1 b) shows the increase of storage space that date need depending on

the number of hiding bits. If we only use one bit per byte to hide data, the required space is 7 times larger than the data. One would have expected to need 8 times more space since 1 bit of data now is represented by 8 bits but I assume it is only 7 due to the compression. The other values are as expected. Using 4 bits, i.e. half a byte double the required space and using 7 bits increases the required space by around 1/7.

4 Future Work

ImagIn encodes within each colors channel of an image the same number of bits. Instead, one might choose the number of hiding bits per channel, e.g. using 5 bits of the red and blue value but only 3 bits of the green. This might lead to better results in terms of quality preservation. For instance, the content an image of a forest might depend stronger on the green channel than the other two. Furthermore, one might split an image up into areas of different smoothness and use different numbers of hiding bits for each. In the image of the ETH building, for instance, we could use only two hiding bits for the sky but 4 bits for the remaining image.

5 Summary

ImagIn encodes files within an image using the least significant bits of a pixel's three bytes. The quality loss strongly depends on the image type. Smooth image show artifacts when using 3 or more bits while noisy images can sometimes even invisibly encode 5 bits per byte. The mean squared error for different image types behaves very equally though. The required storage increase is as one would expect when using more bits to represent them.

6 References

Steganography: http://en.wikipedia.org/wiki/Steganography

Mean squared error: http://en.wikipedia.org/wiki/Mean_squared_error

Scala: http://www.scala-lang.org

Original


1 Hiding bit


2 Hiding bit


3 Hiding bit


4 Hiding bit

5 Hiding bit

6 Hiding bit

7 Hiding bit

Figure 2: The two images for the evaluation: The ETH main building with smooth sky and the noisy forest. Within both images I have encoded bytes using 1 to 7 hiding bits each. The quality gets lost much faster in the smooth image. Even when using 5 bits of each byte of the forest image, it still looks fairly well.