|  |
| --- |
|  |
| Video Steganography: Hiding Video Sequences |
| CSE509: Digital Video Processing |
|  |
| **Aaron Gubrud**  **Jose Trigueros** |
| **25 April 2012** |

Table of Contents

[Introduction 2](#_Toc323078673)

[Problem Statement 2](#_Toc323078674)

[Related Work 2](#_Toc323078675)

[System Overview 3](#_Toc323078676)

[Approach 1 3](#_Toc323078677)

[Approach 2 3](#_Toc323078678)

[Algorithm: Embedding 3](#_Toc323078679)

[Algorithm: Extracting 4](#_Toc323078680)

[Results 5](#_Toc323078681)

[Approach 1 5](#_Toc323078682)

[Approach 2 5](#_Toc323078683)

[Discussion 6](#_Toc323078684)

[Conclusion 6](#_Toc323078685)

[References 6](#_Toc323078686)

[Appendix 6](#_Toc323078687)

# Introduction

The concept of steganography is one that has been around for centuries. Much like encryption its purpose is to take a message and assemble it into a form where - should it be intercepted - the interceptor should not be able to know what the message says. The approaches for encryption and steganography are completely different, however. Whereas encryption takes a given message and performs operations on it such that an interceptor will not be able to read the message, steganography banks on the interceptor not being able to detect the presence of a message at all. Early examples of steganography primarily rely heavily on the transportation medium of the message. One common example is of slave owners in ancient Greek times shaving the head of a slave, tattooing a message on their scalp, allowing the hair on the slave’s scalp to grow back, and then sending them to deliver the message. In the event that they were stopped, nothing would appear suspicious and they would be sent along to complete their task.

In the present, steganography has applications that are more modern. Taking advantage of least significant bits of pixel data in an image file can allow for storage of a message. Two common methods for this type of steganography are applied in the spatial and frequency domains. In the spatial domain, the last few bits of each pixel value within an image can store part of a message. In the frequency domain, a discrete cosine transform can be performed on regularly-sized blocks of a “carrier image”[[1]](#footnote-1) and the coefficient matrix created by the transform creates hiding spots in coefficients that are less important to image structure and quality. These are both simplified explanations but capture the basic principles behind their respective strategies.

When it is said that a message can be hidden within a carrier image, the message itself can be anything from encoded characters in a sequence to entire “seed images”[[2]](#footnote-2). Logically, there is a tradeoff between how much information can be hidden and the visible degradation in image quality of the carrier. This guides choices for both carrier and seed sizes.

# Problem Statement

Since steganographic principles can be applied within digital imaging, it should follow that similar techniques can be applied within digital video. Moreover, it should be possible to hide an entire seed video sequence within a carrier video sequence. Two approaches to this problem will be attempted: one working within the spatial domain and the other working within the frequency domain.

# Related Work

There has been substantial work done in the field of image steganography, but from the research performed there wasn’t very much on video steganography. The work from image steganography could be easily translated to our purposes since we planned on working on a frame by frame basis. The paper *Bit Length Replacement Steganography Based on DCT[[3]](#footnote-3),* provided us with a high level overview of the general process used for hiding information in the DCT domain. Since the paper was not very detailed in the way the information is being hidden, we combined other sources that use the LSB method, in which the seed information is stored only in the LSB values of the carrier image.

# System Overview

Upon coming up with several ideas on how we could make this work for our purposes, we concluded that we should keep a few variables constant if we wanted to come up with a product by the end of the project.

We chose two different formats for the seed and carrier sequences. We used CIF[[4]](#footnote-4) for the carrier image with has a resolution of 352 × 288 and QCIF[[5]](#footnote-5) for the seed sequence, which is a quarter of a CIF frame in terms of resolution. It was also decided that we will use YUV 4:2:0 video sequences, therefore we wouldn’t have to worry about these formats later on.

We ended up working on two methods that are distinct enough where they couldn’t be combined so we chose to use both methods in for comparison metrics.

## Approach 1

## Approach 2

This approach is very similar to the first approach; the contrast lies in the method of selecting where to hide the information, how to select the information to hide, and the ability to have variable input in terms of the amount of the seed image to hide.

### Algorithm: Embedding

A high level description of the algorithm used for embedding the seed sequence is given by the following steps:

1. User will give number *n*, denoting the number of bits to encode per luminance value in the seed frame (since our seed image is in YUV format, we only care about the Y channel when embedding the frames).
2. Obtain the desired number of frames for the seed and carrier sequences.
3. Apply 8x8 2D DCT on the carrier sequence
4. Create an 352 × 288 bit mask, call it key[[6]](#footnote-6)
   1. Bit mask can either be provided by the user or can be created randomly by the algorithm. The only valid values in this matrix are 0 and 1, doing this it statistically guarantee that the matrix will have roughly an equal distributions of 0’s and 1’s.
5. Iterate through the frames of the carrier sequence ( which are now frequency values ) and for each of the frames do the following:
   1. Iterate through the values in the entire DCT carrier sequence with has a one-to-one correspondence with the key. If the current value in the key is 1, then we encode the next n-bits of the seed frame pixel in the LSB[[7]](#footnote-7) of the corresponding carrier value. Continue this until you run out of seed pixels or you reach the end of the key.
6. Once that is completed, compute the 8x8 2D IDCT on the carrier sequence.
7. Output seeded sequence and key used.

Below is a block diagram showing the algorithm:

C:\Users\Jose\Documents\videoSteganography\results\embedding.emf

### Algorithm: Extracting

The extracting algorithm is actually simpler than the embedding as we don’t need to do anything with the carrier frame, we proceed as follows:

1. User will provide *n* which represents the number of bits encoded in each frequency value of the carrier sequence.
2. User will also provide seeded carrier sequence and key.
3. Perform 8x8 2D DCT on seeded carrier sequence
4. For each frame in the sequence do the following:
   1. Iterate through the values of key and carrier, and for every index where key is 1, extract the *n* LSB from the corresponding value in carrier and add it to a buffer.
   2. Accumulate enough bits to create a pixel seed value and store it.
   3. Collect all stored values and create the seed frame.
5. Combine the collection of seed frames, and create the extracted sequence.
6. Output that sequence.

Below is a diagram showing the process that occurs when extracting a sequence:

C:\Users\Jose\Documents\videoSteganography\results\extracting.emf

The idea behind this method is that a user will be able to provide a key or have the algorithm create one, in this way the key will be used to determine the way in which the seed sequence should be encoded in the carrier sequence. In addition, the user can select the precision, to tweak the resulting carrier sequence. The reason that we need to tweak the carrier image and not the seeded image, is because it is more important to make the carrier sequence more similar to the original so that there is less suspicion of data hiding.

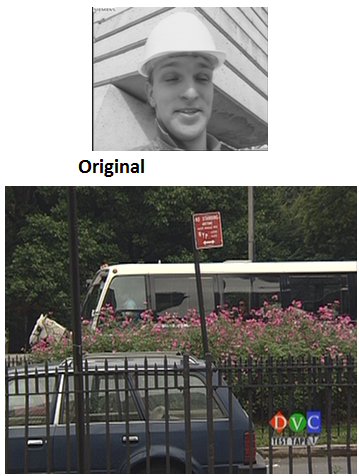
# Results

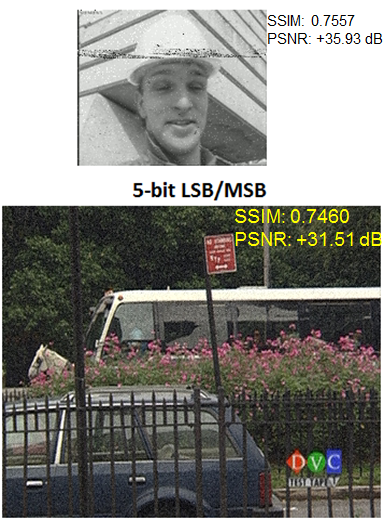
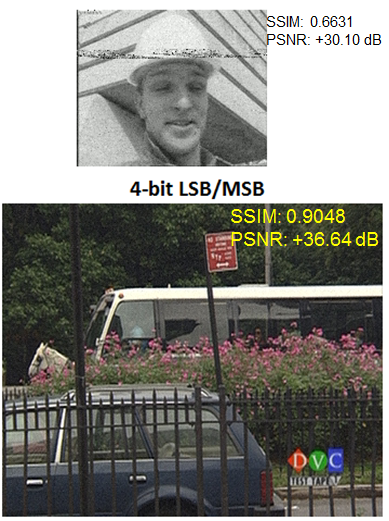
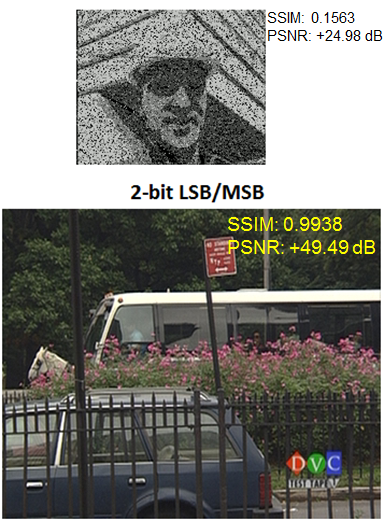
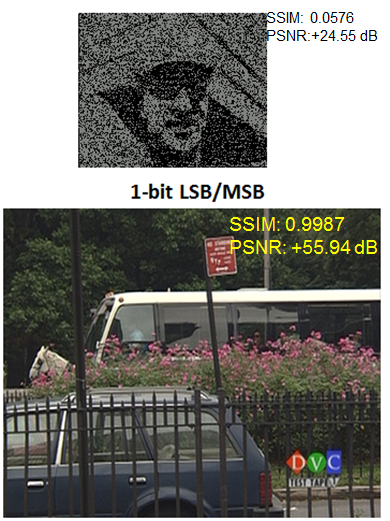
## Approach 1

## Approach 2

After trying different parameters, we obtained the following results using the sequence *bus\_cif.yuv* as the carrier sequence and ­*foreman\_qcif.yuv* as the seed sequence.

These are the results using different precisions given by the user:





From these results it is very apparent that the less we save in the carrier image the better the resulting seeded carrier, but then we lose a lot once we extract the remaining seed. So we try to tweak algorithm with values that will get us something that we can work with, and from this it seems like 4-bit precision is the best we can do without distorting the seeded carrier too much and recovering a useful seed.

These are other results we obtained using two other CIF sequences that contain different patterns, the SSIM and PSNR values are averaged over 5 frames.

<insert results>

# Discussion

# Conclusion

Initially our expectations were too high, we wanted a way to store an seed of the same size as the carrier, we soon learned that something must be sacrificed. We either had to sacrifice the length of seed sequence, store a smaller resolution, or lose bit precision. We ended up giving up the bit precision and choosing smaller resolution for the seed image, this would guarantee that we have enough space to manipulate both sequences adequately and be able to store the data efficiently.

In order to keep our algorithms consistent and simplified we also chose specific formats that would allow us to show that this can be done, and if it can be done with these restrictions then this problem can be built upon and expanded to different dimensions.

# References

[1]

“YUV Sequences.” [Online]. Available: http://trace.eas.asu.edu/yuv/index.html. [Accessed: 25-Apr-2012].

[2]

“www.FOURCC.org - Video Codecs and Pixel Formats.” [Online]. Available: http://www.fourcc.org/. [Accessed: 25-Apr-2012].

[3]

D. Yu, “Convert YUV CIF 4:2:0 video file to image files - File Exchange - MATLAB Central.” [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/6318. [Accessed: 25-Apr-2012].

[4]

Z. Wang, “The SSIM Index for Image Quality Assessment.” [Online]. Available: http://www.cns.nyu.edu/~lcv/ssim/. [Accessed: 25-Apr-2012].

[5]

J. T. McCarthy, “PSNR: PSNR(A,B) - File Exchange - MATLAB Central.” [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/135-psnr/content/PSNR.m. [Accessed: 25-Apr-2012].

[6]

K. B. . Kumar, K. B. Raja, R. K. Chhotaray, and S. Pattanaik, “Bit Length Replacement Steganography based on DCT Coefficients,” *International Journal of Engineering Science and Technology*, vol. 2, no. 8, pp. 3561–3570, 2010.

# Appendix

1. Carrier image: image in which a message will be stored. [↑](#footnote-ref-1)
2. Seed image: image which will be hidden within the carrier image. [↑](#footnote-ref-2)
3. Discrete Cosine Transform [↑](#footnote-ref-3)
4. Common Intermediate Format [↑](#footnote-ref-4)
5. Quarter CIF [↑](#footnote-ref-5)
6. Key: term used to describe the bit mask created to encode and decode the carrier sequence [↑](#footnote-ref-6)
7. LSB: Least Significant Bit [↑](#footnote-ref-7)