

Image Steganography- CONT.

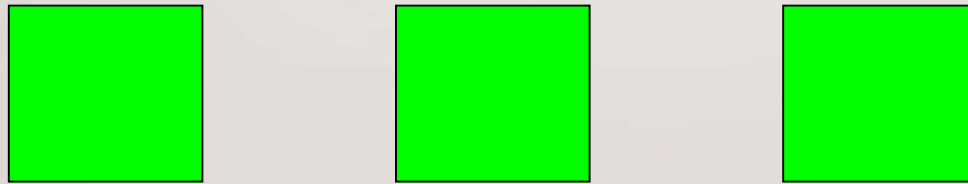
- An image is represented as an $N \times M$ (in case of greyscale images) or $N \times M \times 3$ (in case of colour images) matrix in memory, with each entry representing the intensity value of a pixel.
- In image steganography, a message is embedded into an image by altering the values of some pixels, which are chosen by an encryption algorithm.
- The recipient of the image must be aware of the same algorithm in order to know which pixels he or she must select to extract the message.

LEAST-SIGNIFICANT BIT (LSB) EMBEDDING - TERMINOLOGY

- *Message* = the secret information we want to hide
- *Cover image* = image used to hide the message in
- *Stego-image* = the cover image with the message embedded

LEAST-SIGNIFICANT BIT (LSB) - CONCEPT

- Which color is different?



- In (R,G,B) left and right are (0,255,0)
- Center one is (0,254,0)
- We can use the LSB to hold info, since it looks the same either way!

LSB 24-BIT BITMAPS

- In 24-bit bmps, each pixel represented by 3 bytes (RGB)
- Use lsb of each byte to hold a bit of message
- Example: LSB 24-bit Bitmaps

Message = 'f' = $0110\ 0110_2$

Cover Image:

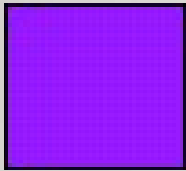
FF FF FF 00 00 00 FF FF ...

Stego-image:

FE FF FF 00 00 01 FF FE ...

Simple Example

Original RGB Pixel



[150 25 255]

MSB LSB

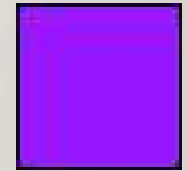
1111 1111 ← 1

0001 1001 ← 0

1001 0110 ← 1

1010101
1101110
1101001
1110110
1100101
1110010
1110011
1101001
1110100
1111001
0100000
1101111
1100110
0100000
1010101
1101100
1110011
1110100
1100101
1110010

Modified RGB Pixel



[151 24 255]

1111 1111

0001 1000

1001 0111

Binary ('University of Tikrit');

LSB - programming- Grayscale

- The pseudo-code below can be used to explain the processing to embed a text message in a grayscale image by replacing the LSB of each pixel:

```
pic=cover image
msg=secret message
n=number of chars in msg
for i=1 to n
  get char from msg
  for each bit in char
    get a pixel from pic
    if the bit=1
      insert a 1 in the least significant bit of the pixel
    else
      insert a 0 in the least significant bit of the pixel
    replace the pixel in pic
  end for
end for
```


LSB - programming- Color CONT.

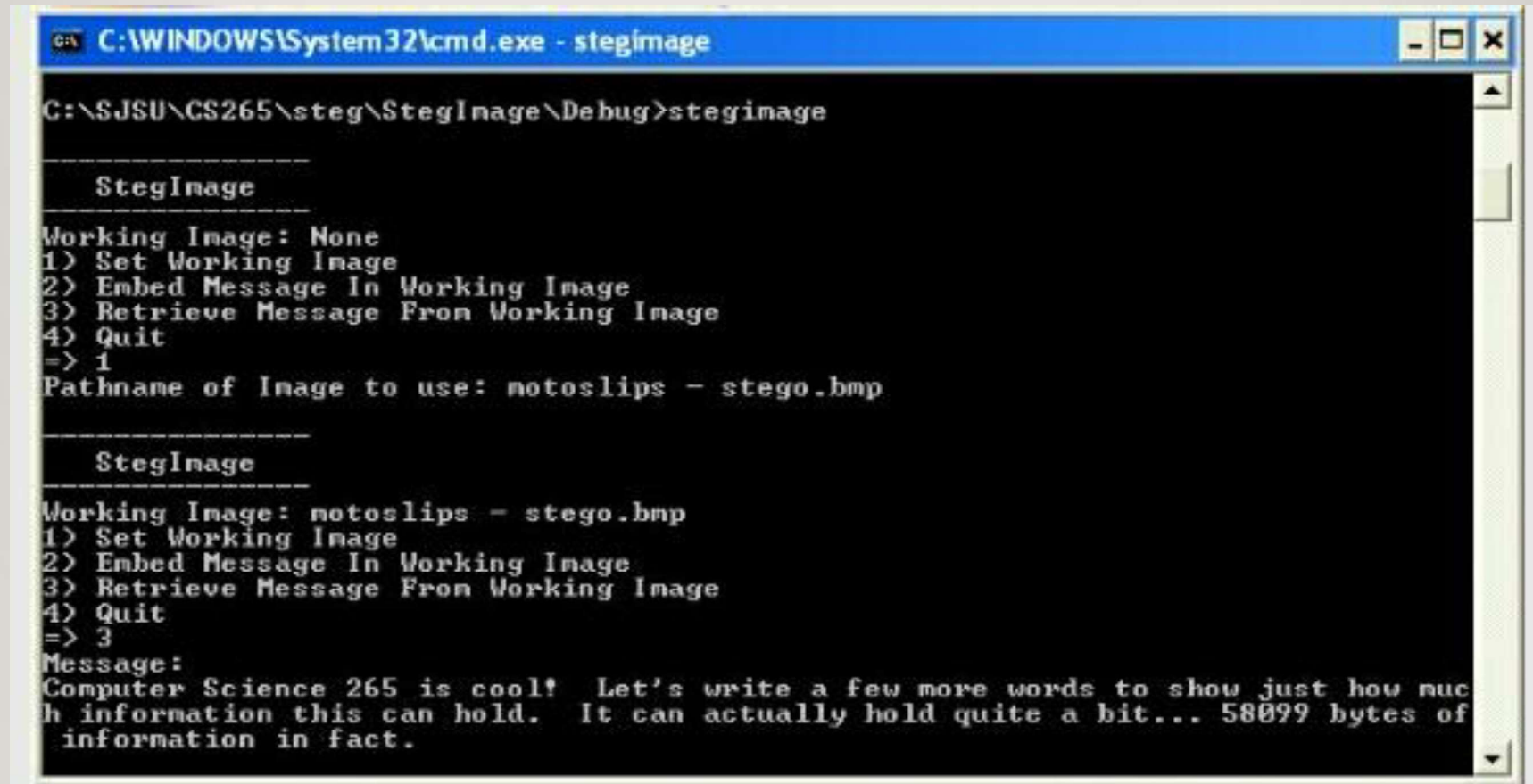
- The pseudo-code below can be used to explain a simple LSB insertion algorithm used with 24-bit images:

```
pic=cover image
msg=secret message
n=number of chars in msg
for i=1 to n
  get char from msg
  for each 3 bits in char
    get a pixel from pic
    get the red value of the pixel
```

LSB - programming CONT.

```
if the first bit=1
  insert a 1 in the least significant bit of the red value
else
  insert a 0 in the least significant bit of the red value
  get the green value of the pixel
if the second bit=1
  insert a 1 in the least significant bit of the green
  value
else
  insert a 0 in the least significant bit of the green
  value
  get the blue value of the pixel
if the third bit=1
  insert a 1 in the least significant bit of the blue
  value
else
  insert a 0 in the least significant bit of the blue
  value
  replace the value in pic
end for
end for
```


LSB - Implementation...CONT.



```
C:\WINDOWS\System32\cmd.exe - stegimage
C:\SJSU\CS265\steg\StegImage\Debug>stegimage

-----
StegImage
-----
Working Image: None
1) Set Working Image
2) Embed Message In Working Image
3) Retrieve Message From Working Image
4) Quit
=> 1
Pathname of Image to use: notoslips - stego.bmp

-----
StegImage
-----
Working Image: notoslips - stego.bmp
1) Set Working Image
2) Embed Message In Working Image
3) Retrieve Message From Working Image
4) Quit
=> 3
Message:
Computer Science 265 is cool! Let's write a few more words to show just how much
information this can hold. It can actually hold quite a bit... 58099 bytes of
information in fact.
```

8 bits

- Don't hold direct color values
- Do hold offsets into a palette
- Can't just change lsb, because adjacent colors in palette may not be similar