**David Hazall-Farrell**

**CS 132 Lab 6**

**4 hours**

**The ImageHider class was a bit difficult to understand and set up. It took me a long time to figure out what I was doing for it.**

**ImageClearBits:**

**package** CS132Images;

**import** squint.SImage;

**public** **class** ImageClearBits **implements** ImageTransformer {

**private** **int** divisionFactor;

// Constructor that accepts a power and calculates divisionFactor as 2^power

**public** ImageClearBits(**int** power) {

**this**.divisionFactor = (**int**) Math.*pow*(2, power);

}

// Default constructor with a divisionFactor based on 2^3

**public** ImageClearBits() {

**this**(3); // default power of 3, so divisionFactor is 8

}

@Override

**public** SImage transform(SImage picture) {

**return** clearBits(picture);

}

// Applies bit clearing to the image by processing each channel separately

**private** SImage clearBits(SImage si) {

**int**[][] newReds = clearBitsSingleChannel(si.getRedPixelArray());

**int**[][] newGreens = clearBitsSingleChannel(si.getGreenPixelArray());

**int**[][] newBlues = clearBitsSingleChannel(si.getBluePixelArray());

**return** **new** SImage(newReds, newGreens, newBlues);

}

// Clears bits for a single color channel

**private** **int**[][] clearBitsSingleChannel(**int**[][] pixelArray) {

**int** rows = pixelArray.length;

**int** columns = pixelArray[0].length;

**int**[][] answer = **new** **int**[rows][columns];

**for** (**int** r = 0; r < rows; r++) {

**for** (**int** c = 0; c < columns; c++) {

answer[r][c] = clearBitsForPixel(pixelArray[r][c]);

}

}

**return** answer;

}

// Clears bits for a single pixel by applying the "/1000\*1000" approach

**private** **int** clearBitsForPixel(**int** pixelValue) {

**return** (pixelValue / divisionFactor) \* divisionFactor;

}

}

**ImageStegExtractor:**

**package CS132Images;**

**import squint.SImage;**

**public class ImageStegExtractor implements ImageTransformer {**

**private int extractionFactor;**

**// Constructor that accepts a power and calculates extractionFactor as 2^power**

**public ImageStegExtractor(int power) {**

**this.extractionFactor = (int) Math.*pow*(2, power);**

**}**

**public ImageStegExtractor() {**

**this(4); // default to extracting 4 bits**

**}**

**@Override**

**public SImage transform(SImage picture) {**

**return extractBits(picture);**

**}**

**private SImage extractBits(SImage si) {**

**int[][] newReds = extractBitsSingleChannel(si.getRedPixelArray());**

**int[][] newGreens = extractBitsSingleChannel(si.getGreenPixelArray());**

**int[][] newBlues = extractBitsSingleChannel(si.getBluePixelArray());**

**return new SImage(newReds, newGreens, newBlues);**

**}**

**// Extracts bits for a single color channel**

**private int[][] extractBitsSingleChannel(int[][] pixelArray) {**

**int rows = pixelArray.length;**

**int columns = pixelArray[0].length;**

**int[][] answer = new int[rows][columns];**

**for (int r = 0; r < rows; r++) {**

**for (int c = 0; c < columns; c++) {**

**answer[r][c] = extractBitsForPixel(pixelArray[r][c]);**

**}**

**}**

**return answer;**

**}**

**extractionFactor**

**private int extractBitsForPixel(int pixelValue) {**

**return pixelValue % extractionFactor;**

**}**

**}**

**ImageHider:**

/\*\*

\* ImageToClearBits - an ImageTransformer that reduces the source image to

\* one bit per color channel (hence eight colors overall)

\*

\* @author David Levine

\* @version May 28, 2009

\*/

package CS132Images;

import squint.SImage;

public class ImageHider {

private int divisionFactor;

public ImageHider() {

this(1);

}

public ImageHider(int bitsToClear) {

divisionFactor = 2;

}

public SImage transform(SImage cover, SImage secret) {

return hide(cover, secret);

}

private SImage hide(SImage cover, SImage secret) {

int[][] newReds = hideSingleChannel(cover.getRedPixelArray(), secret.getRedPixelArray());

int[][] newGreens = hideSingleChannel(cover.getGreenPixelArray(), secret.getGreenPixelArray());

int[][] newBlues = hideSingleChannel(cover.getBluePixelArray(), secret.getBluePixelArray());

return new SImage(newReds, newGreens, newBlues);

}

private int[][] hideSingleChannel(int[][] cover, int[][] secret) {

int rows = cover.length;

int columns = cover[0].length;

int[][] result = new int[rows][columns];

for (int r = 0; r < rows; r++) {

for (int c = 0; c < columns; c++) {

result[r][c] = hideBitsInPixel(cover[r][c], secret[r][c]);

}

}

return result;

}

private int hideBitsInPixel(int coverPixel, int secretPixel) {

int clearedCoverPixel = coverPixel & ~((1 << divisionFactor) - 1);

int secretBits = (secretPixel >> (8 - divisionFactor)) & ((1 << divisionFactor) - 1);

// Combine the cleared cover pixel with the secret bits in the LSBs

return clearedCoverPixel | secretBits;

}

}

2. I think the image degrades noticeably at around the 5th sequence. It starts getting darker and unrecognizable. At around the 7th sequence the image is hardly the same at that point.

5. When extracting only two bits, the image turns extremely pixelated but extremely colorful. Lots of white is shown as well, while very few parts of the original image remain.