Учреждение образования

«Белорусский государственный технологический университет»

**Отчет по лабораторной работе №12**

по дисциплине: «Защита информации и надежность информационных систем»

Исполнитель:

Воликов Д. А., ФИТ 4-7

Руководитель:

Асс. Савельева М. Г.

Минск 2024

# Лабораторная работа №12. Исследование стеганографического метода на основе преобразования наименее значащих битов

**Цель**: изучение стеганографического метода встраивания\*/извлечения тайной информации с использованием электронного файла-контейнера на основе преобразования наименее значащих битов (НЗБ), приобретение практических навыков программной реализации данного метода.

**Задачи**:

1. Закрепить теоретические знания из области стеганографического преобразования информации, моделирования стеганосистем, классификации и сущности методов цифровой стеганографии.
2. Изучить алгоритм встраивания/извлечения тайной информации на основе метода НЗБ (LSB – Least Significant Bit), получить опыт практической реализации метода.
3. Разработать приложение для реализации алгоритма встраивания/извлечения тайной информации с использованием электронного файла-контейнера на основе метода НЗБ.
4. Познакомиться с методиками оценки стеганографической стойкости метода НЗБ.
5. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

## Теоретические сведения

**Определение 1**. Стеганографическая система (stegosystem, стегосистема или стеганосистема – в русскоязычной тематической литературе используются оба сокращения) – совокупность средств и методов, которые используются для формирования скрытого канала передачи (или хранения) информации.

При этом скрытый канал организуется на базе и внутри открытого канала с использованием особенностей восприятия информации. «Скрытость» канала передачи тайной информации отличает стеганографию от криптографии: в первом случае тайной является сам факт наличия канала (передачи информации).

**Определение 2**. Абстрактно стеганографическая система обычно определяется как некоторое множество отображений одного пространства (множества возможных сообщений М) в другое пространство (множество возможных стеганосообщений S), и наоборот.

Определение 3. Стеганографической системой *∑* будем называть совокупность сообщений *M*, контейнеров *C*, ключей *K*, стеганосообщений (заполненных контейнеров) *S* и преобразований (прямого *F* и обратного *F*–1), которые их связывают:

*∑* = (*M*, *C*, *K*, *S*, *F*, *F*–1).

Метод наименьшего значащего бита. Большинство исследований в предметной области посвящено использованию в качестве стеганоконтейнеров изображений (текст также можно рассматривать как изображение).

Метод НЗБ основывается на ограниченных способностях зрения или слуха человека, вследствие чего людям тяжело различать незначительные вариации цвета или звука. Рассмотрим это на примере 24-битного растрового RGB-изображения. Как известно, каждая точка кодируется тремя байтами. Каждый байт определяет интенсивность красного (Red), зеленого (Green) и синего (Blue) цветов. Совокупность интенсивностей цвета в каждом из трех каналов определяет оттенок пикселя.

Младшие биты (выделены бледным, справа) дают незначительный «вклад» в изображение по сравнению со старшими.

Замена одного или даже нескольких младших битов для человеческого глаза будет почти незаметна, поскольку реально человек может различать около полторы сотни цветовых оттенков.

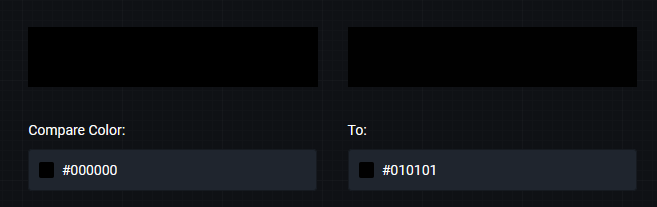


Рисунок 1 – Разница цветов при изменении одного бита в каждом канале

Одним из простейших и понятных для решения наших задач является формат BMP (BitMaP) – одна из форм представления растровой графики. Изображение представляется в виде матрицы пикселей, где каждая точка характеризуется тремя параметрами: *x*-координатой, *y*-координатой и цветовым кодом на основе RGB-модели. Все операции графического ввода-вывода на экран монитора (принтер и на некоторые другие устройства) в конечном итоге осуществляются в этом формате.

Другим из растровых форматов используемых в стеганографии контейнеров является формат PNG (Portable Network Graphics). По качеству цветового отображения данный формат превосходит JPEG (Joint Photographic Experts Group) и GIF (Graphics Interchange Format), но размер файла будет на 30–40% больше.

Следует соотносить объем встроенного сообщения *V*M с объемом *V*C используемого контейнера. Например, если размер изображения 500×500 =  
250 000 пикселей, то с учетом используемой 3-цветовой модели имеем 750 000 единиц цветовых координат. Если мы планируем модифицировать только самые младшие биты всех цветовых каналов матрицы, то максимальный объем осаждаемого сообщения (*V*Mmax) не должен превышать 750 000 битов.

Как вообще можно понять, что в изображении имеется спрятанное сообщение? Для этого берут незначащий бит в каждом канале пикселя и строят на его основе новое изображение.

Для примера возьмём следующее изображение.

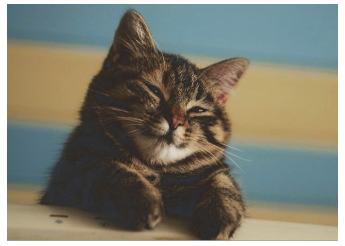


Рисунок 2 – Изображение-контейнер

Далее построим изображение следующим образом: каждый канал пикселя будет равняться 255, если последний бит равен 1, иначе 0. Таким образом получим результат, который отображён на рисунке ниже.

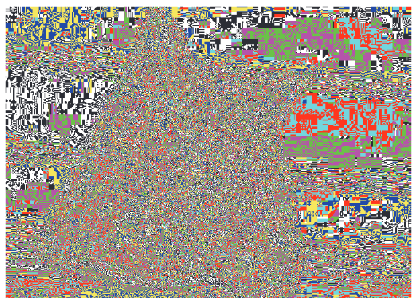


Рисунок 3 – Цветовое отображение младших разрядов «пустого» контейнера

Если же вставить в это изображение некое сообщение на основе алгоритма НЗБ, то мы получим следующий результат.

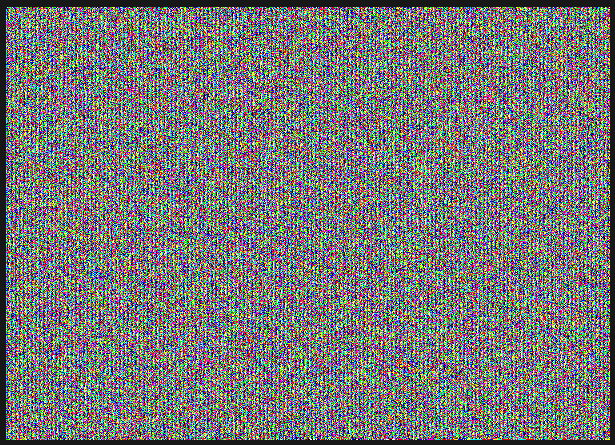


Рисунок 4 – Цветовое отображение младших разрядов «непустого» контейнера

Как можно заметить, разница с рисунком 3 ощутима. Здесь мы уже не видим очертания животного, полосок на фоне и т.п. Таким образом мы доказали, что изображение хранит в себе скрытое сообщение.

## Практическое задание

По условию лабораторной работы следовало разработать собственное приложение, в котором должен быть реализован метод НЗБ. При этом, реализовать 2 варианта размещения битов по контейнеру. В моём случае – размещение битов сообщения по чётным и нечётным пикселям изображения-контейнера.

Возьмём следующее изображение для роли контейнера:



Рисунок 5 – Изображение-контейнер

В качестве сообщения сгенерируем некий случайный набор текста. Сохраним его в файл для дальнейшего чтения его программой. Перед заполнением изображения сообщением получим его цветовое представление наименьшего значащего бита.

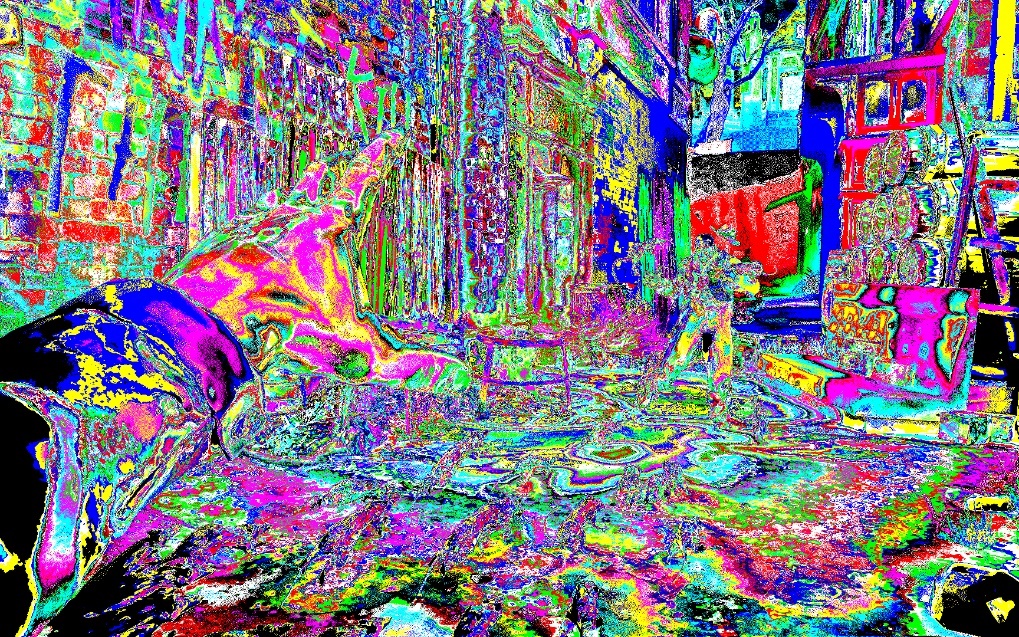


Рисунок 6 – Цветовое представление наименьшего значащего бита контейнера

Рассмотрим на основе распределения сообщения по чётным пикселям. Вставим исходное сообщение в пустой контейнер и проанализируем результат после выполнения работы алгоритма.

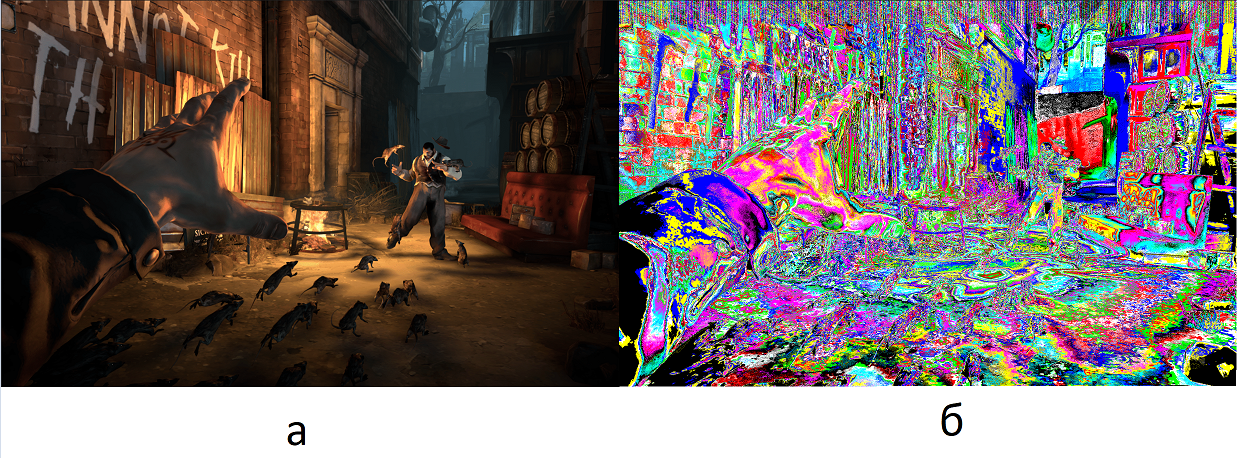


Рисунок 7 – Контейнер после вставки сообщения по чётным пикселям:  
а – обычный вид изображения, б – цветовое представление наименьшего значащего бита

Если сравнить рисунок 5 с рисунком 7а, то разница для человеческого глаза абсолютно незаметна. Но если сравним рисунок 6 с рисунком 7б, то увидим, что верх контейнера имеет некую «зашумленность», что говорит о том, что здесь спрятано исходное сообщение.

Чтобы получить сообщение, находящееся внутри изображения, программа использует метаданные изображения на основе Exif. Таким образом, количество байт сообщения находятся в теге ImageNumber (что по себе логично, ибо прячется число байтов сообщения), а способ распределения битов в теге Copyright (True – распределение по чётным пикселям, False – по нечётным). Таким образом, при получении изображения, извлекаются эти значения, и на их основе считываются биты, которые потом преобразуются в символы.

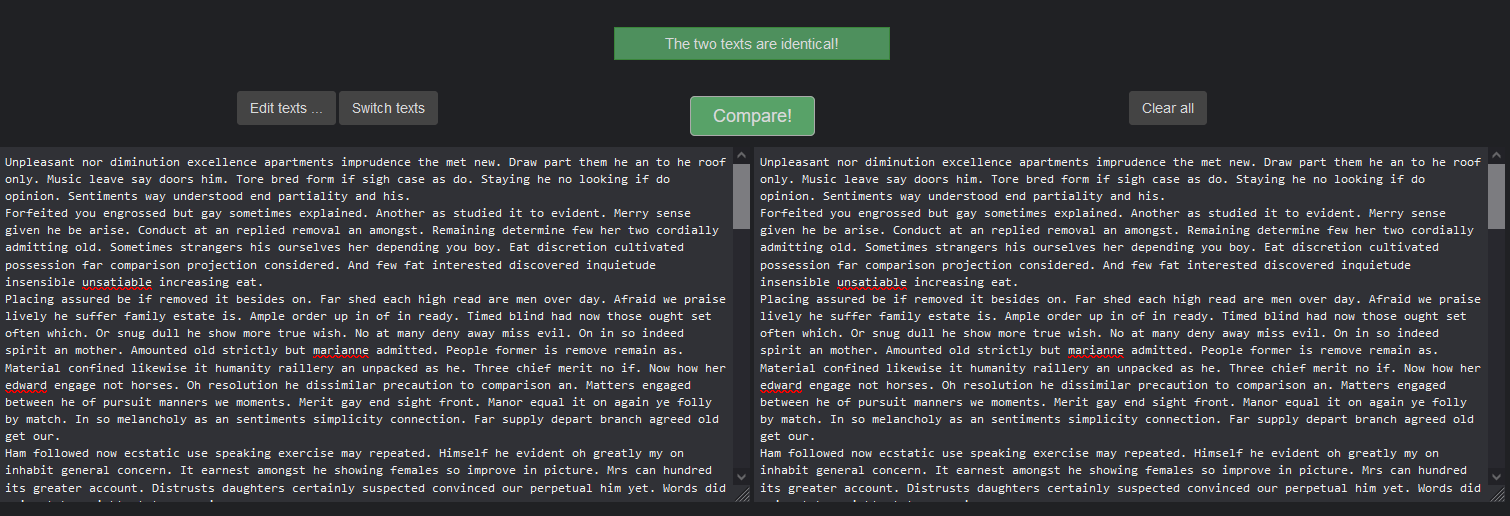


Рисунок 8 – Результат сравнения исходного сообщения с полученным из изображения

Как можно заметить на рисунке выше, сравнение показало положительный результат. Таким образом делаем вывод о том, что алгоритм работает исправно, а также исправно читает спрятанный текст.

Рассмотрим на основе распределения сообщения по нечётным пикселям. Изображение-контейнер всё тоже. Увеличим немного исходное сообщение и вставим его в пустой контейнер и проанализируем результат после выполнения работы алгоритма.



Рисунок 9 – Контейнер после вставки сообщения по нечётным пикселям:  
а – обычный вид изображения, б – цветовое представление наименьшего значащего бита

При сравнение исходного изображения с изображением с сообщением внутри не видно никаких дефектов. Однако если сравнить цветовые представления наименьших значащих битов, то можно заметить, что «зашумлённости» появилась и её стало больше (ибо увеличено исходное сообщение).

Получим скрытое сообщение из изображения и сравним его с исходным, дабы убедиться в том, что они считывается правильно.

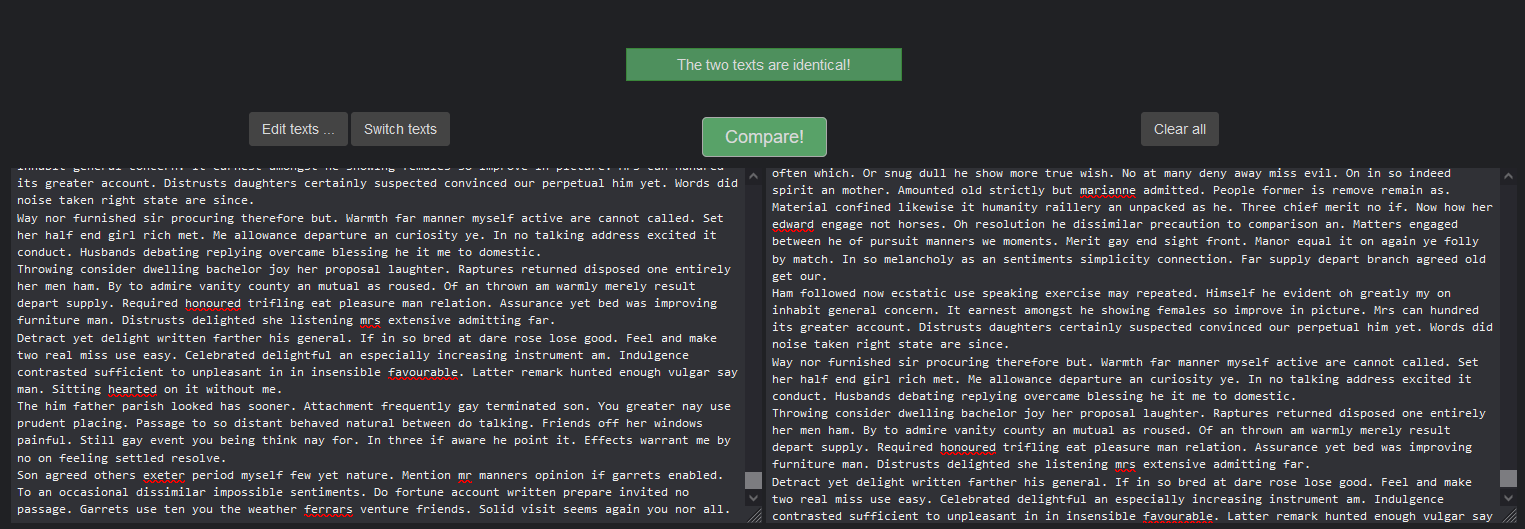


Рисунок 10 – Результат сравнения исходного сообщения с полученным из изображения

Результат сравнения сообщений дал положительный результат. Это значит, что два способа распределения битов сообщения по контейнеру работают так, что сообщение отлично скрывается в изображении без артефактов на нём, а также правильно считывается.

Вывод: в ходе этой лабораторной работы я изучил теоретические основы понятия стеганографии на основе наименьшего значащего бита. Разработал программу с реализацией данного алгоритма. За основу распределения битов сообщения по контейнеру взяты два метода: распределение по чётным пикселям, а также распределение по нечётным. Протестировал программу на различных входных параметрах и убедился в том, что она работает правильно.

# Листинг «Лабораторная работа №12»

using System.Text;

using SixLabors.ImageSharp;

using SixLabors.ImageSharp.Metadata.Profiles.Exif;

namespace Lab12.LSB

{

public static class LSB

{

public static void Encrypt(string filename, string message, bool even = true)

{

var messageBytes = Encoding.UTF8.GetBytes(message);

var messageBits = string.Join("", messageBytes.Select(b => Convert.ToString(b, 2).PadLeft(8, '0')));

var (pixels, width, height) = ImageProcess.ImageProcess.GetImageBytes(filename);

if (pixels.Length \* 3 / 2 < messageBits.Length)

{

throw new Exception("Too small image for your message");

}

var counter = 0;

for (int i = even ? 0 : 1; i < pixels.Length; i += 2)

{

pixels[i].R = GetEncryptedPixelChannelByte(pixels[i].R, messageBits[counter++]);

if (counter == messageBytes.Length \* 8) break;

pixels[i].G = GetEncryptedPixelChannelByte(pixels[i].G, messageBits[counter++]);

if (counter == messageBytes.Length \* 8) break;

pixels[i].B = GetEncryptedPixelChannelByte(pixels[i].B, messageBits[counter++]);

if (counter == messageBytes.Length \* 8) break;

}

var fileInfo = new FileInfo(filename);

var filenameWOExtension = Path.GetFileNameWithoutExtension(filename);

var newFilename = fileInfo.DirectoryName + "\\" + filenameWOExtension + "-encrypted.png";

ExifProfile profile = new();

profile.SetValue(ExifTag.ImageNumber, (uint)messageBytes.Length);

profile.SetValue(ExifTag.Copyright, even.ToString());

ImageProcess.ImageProcess.SaveImage(pixels, width, height, newFilename, profile);

}

public static string Decrypt(string filename)

{

var (pixels, \_, \_) = ImageProcess.ImageProcess.GetImageBytes(filename);

var profile = ImageProcess.ImageProcess.GetExifProfile(filename);

IExifValue<uint>? messageLengthValue = null;

IExifValue<string>? isEvenValue = null;

profile?.TryGetValue(ExifTag.ImageNumber, out messageLengthValue);

profile?.TryGetValue(ExifTag.Copyright, out isEvenValue);

uint messageLength = messageLengthValue == null ? 0 : (uint)messageLengthValue.GetValue()!;

bool isEven = isEvenValue == null || (string)isEvenValue!.GetValue()! == "True";

if (messageLength == 0) return "There is no any message";

var counter = 0;

var binaryData = "";

byte[] bytes = new byte[messageLength];

for (int i = isEven ? 0 : 1; i < pixels.Length; i += 2) {

binaryData += pixels[i].R & 1;

if (++counter == messageLength \* 8) break;

binaryData += pixels[i].G & 1;

if (++counter == messageLength \* 8) break;

binaryData += pixels[i].B & 1;

if (++counter == messageLength \* 8) break;

}

for(int i = 0; i < messageLength; i++)

{

bytes[i] = Convert.ToByte(binaryData.Substring(8 \* i, 8), 2);

}

return Encoding.UTF8.GetString(bytes);

}

private static byte GetEncryptedPixelChannelByte(byte pixelChannelByte, char messageBit)

{

int lastChannelBit = pixelChannelByte & 1;

if (lastChannelBit.ToString() == messageBit.ToString()) {

return pixelChannelByte;

}

return (byte)(pixelChannelByte ^ 1);

}

}

}

Листинг 1 – Класс «LSB»

using SixLabors.ImageSharp;

using SixLabors.ImageSharp.Metadata.Profiles.Exif;

using SixLabors.ImageSharp.PixelFormats;

namespace Lab12.ImageProcess

{

public class ImageProcess

{

public static (Rgb24[], int, int) GetImageBytes(string filename)

{

if (!File.Exists(filename))

{

throw new Exception("No such file " + filename + "\nCheck the file path and try again");

}

using Image<Rgb24> image = Image.Load<Rgb24>(filename);

var pixels = new Rgb24[image.Width \* image.Height];

image.CopyPixelDataTo(pixels);

return (pixels, image.Width, image.Height);

}

// channel 0 - red, 1 - green, 2 - blue

public static void GetLSBImageChannel(string filename, int channel)

{

var (pixels, width, height) = GetImageBytes(filename);

for (int i = 0; i < pixels.Length; i++)

{

pixels[i] = GetPixelByChannel(pixels[i], channel);

}

using Image image = Image.LoadPixelData<Rgb24>(pixels, width, height);

var fileInfo = new FileInfo(filename);

var filenameWOExtension = Path.GetFileNameWithoutExtension(filename);

var newFilename = fileInfo.DirectoryName + "\\" + filenameWOExtension;

newFilename += channel == 0 ? "-lsb\_red.png" : channel == 1 ? "-lsb\_green.png" : "-lsb\_blue.png";

if (File.Exists(newFilename)) File.Delete(newFilename);

image.SaveAsPng(newFilename);

}

public static void GetLSBImage(string filename)

{

var (pixels, width, height) = GetImageBytes(filename);

for (int i = 0; i < pixels.Length; i++)

{

pixels[i].R = (pixels[i].R & 1) == 1 ? (byte)255 : (byte)0;

pixels[i].G = (pixels[i].G & 1) == 1 ? (byte)255 : (byte)0;

pixels[i].B = (pixels[i].B & 1) == 1 ? (byte)255 : (byte)0;

}

using Image image = Image.LoadPixelData<Rgb24>(pixels, width, height);

var fileInfo = new FileInfo(filename);

var filenameWOExtension = Path.GetFileNameWithoutExtension(filename);

var newFilename = fileInfo.DirectoryName + "\\" + filenameWOExtension + "-full\_lsb.png";

if (File.Exists(newFilename)) File.Delete(newFilename);

image.SaveAsPng(newFilename);

}

public static ExifProfile? GetExifProfile(string filename)

{

if (!File.Exists(filename))

{

throw new Exception("No such file " + filename + "\nCheck the file path and try again");

}

using Image<Rgb24> image = Image.Load<Rgb24>(filename);

return image.Metadata.ExifProfile;

}

public static void SaveImage(Rgb24[] pixels, int width, int height, string filename, ExifProfile profile)

{

if (File.Exists(filename)) File.Delete(filename);

using Image newImage = Image.LoadPixelData<Rgb24>(pixels, width, height);

newImage.Metadata.ExifProfile = profile;

newImage.SaveAsPng(filename);

}

private static Rgb24 GetPixelByChannel(Rgb24 pixel, int channel)

{

if (channel == 0)

{

return (pixel.R & 1) == 1 ? new Rgb24(0, 0, 0) : new Rgb24(255, 255, 255);

}

if (channel == 1)

{

return (pixel.G & 1) == 1 ? new Rgb24(0, 0, 0) : new Rgb24(255, 255, 255);

}

if (channel == 2)

{

return (pixel.B & 1) == 1 ? new Rgb24(0, 0, 0) : new Rgb24(255, 255, 255);

}

return pixel;

}

}

}

Листинг 2 – Класс «ImageProcess»