МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ЭКОНОМИКИ, СТАТИСТИКИ И ИНФОРМАТИКИ (МЭСИ)

Курсовая работа

по .NET-программированию

На тему: «Построение трасс с огибанием препятствий»

Выполнил студент  
группы ДКО-122б  
Вишняков Ф.Н.

Проверил: доцент  
Чапкин Н. С.

Москва, 2014

Оглавление

[Постановка цели 3](#_Toc407399473)

[Описание задач 3](#_Toc407399474)

[Определение свободных и непроходимых областей 3](#_Toc407399475)

[Метод Оцу 4](#_Toc407399476)

[Построение полного графа и нахождение весов и координат каждого ребра 5](#_Toc407399477)

[Волновой поиск (алгоритм волновой трассировки, алгоритм Ли) 5](#_Toc407399478)

[Построение минимального остовного дерева 6](#_Toc407399479)

[Алгоритм Прима 6](#_Toc407399480)

[Диаграммы классов 7](#_Toc407399481)

[Описание основных методов 8](#_Toc407399482)

[MainForm 8](#_Toc407399483)

[TrackFinder 9](#_Toc407399484)

[Листинг программы 10](#_Toc407399485)

[Tracks - Main.cs 10](#_Toc407399486)

[GUI – Program.cs 17](#_Toc407399487)

[GUI – MainForm.cs 18](#_Toc407399488)

# Постановка цели

Разработать и реализовать алгоритм построения минимального остовного дерева трасс, соединяющих заданные точки карты с огибанием препятствий. Препятствия описаны выпуклыми многоугольниками, и огибающая должна проходить через одну или несколько вершин многоугольника по его граням. Продемонстрировать работу алгоритма на плане города (многоугольники с преимущественно прямыми углами) и картах местности (выпуклые многоугольники, описывающие озёра).

Для выполнения поставленной цели нам потребуется:

* Загрузить изображение
* Определить свободные и непроходимые области
* Задать точки на карте
* Рассчитать расстояния и пути от всех точек к каждой другой, тем самым получив полный граф
* Построить минимальное остовное дерево полученного связанного неориентированного весового графа
* Вывести результат

# Описание задач

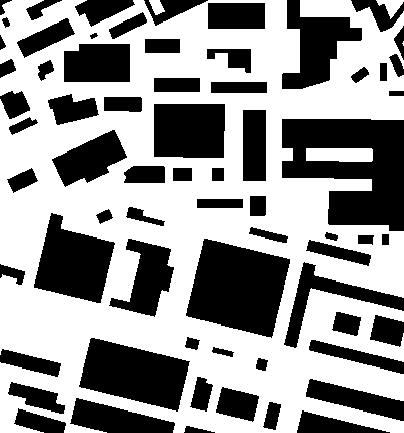
Загрузка изображения, задание точек и вывод результата – задачи тривиальные, мы не будем останавливаться на них.

## Определение свободных и непроходимых областей

Для выполнения этой задачи была выбрана бинаризация изображения. Бинаризация в нашем случае – это приведение цветного изображения к черно-белому с сохранением требуемой информации. Белый цвет пикселя получившегося изображения означает его проходимость, черный – обратное.

В программе мы будем иметь дело с двумя типами изображений – городская карта и местность с водой. Для их обработки нам потребуется два критерия. Для обнаружения препятствий в городе используем метод Оцу, для воды же понадобится лишь найти «синие» пиксели.

Пример распознавания городской карты:

Пример распознавания водной карты:





### Метод Оцу

Метод Оцу — это алгоритм вычисления порога бинаризации для полутонового изображения, используемый в области компьютерного распознавания образов и обработки изображений.

Алгоритм позволяет разделить пиксели двух классов («полезные» и «фоновые»), рассчитывая такой порог, чтобы внутриклассовая дисперсия была минимальной.

Метод Оцу ищет порог, уменьшающий дисперсию внутри класса, которая определяется как взвешенная сумма дисперсий двух классов:

\sigma^2_w(t)=\omega_1(t)\sigma^2_1(t)+\omega_2(t)\sigma^2_2(t),

где веса ωi — это вероятности двух классов, разделенных порогом t, σ2i — дисперсия этих классов.

Оцу показал, что минимизация дисперсии внутри класса равносильна максимизации дисперсии между классами:

\sigma^2_b(t)=\sigma^2-\sigma^2_w(t)=\omega_1(t)\omega_2(t)\left[\mu_1(t)-\mu_2(t)\right]^2

которая выражается в терминах вероятности ωi и среднего арифметического класса μi, которое, в свою очередь, может обновляться итеративно. Эта идея привела к эффективному алгоритму.

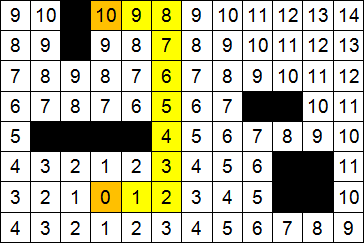
Алгоритм:

1. Вычислить гистограмму и вероятность для каждого уровня интенсивности.
2. Вычислить начальные значения для ωi(0) и μi(0).
3. Для каждого значения порога от t = 1 .. до максимальной интенсивности:
   1. Обновляем \omega_i и \mu_i
   2. Вычисляем σ2b(t).
   3. Если σb(t) больше, чем имеющееся, то запоминаем σb и значение порога t.
4. Искомый порог соответствует максимуму σ2b(t).

## Построение полного графа и нахождение весов и координат каждого ребра

После получения бинаризованного изображения, можно приступить к расчету путей. Для их нахождения воспользуемся волновым поиском.

### Волновой поиск (алгоритм волновой трассировки, алгоритм Ли)



Это алгоритм поиска кратчайшего пути на графе.

Работа алгоритма включает в себя три этапа: инициализацию, распространение волны и восстановление пути. Первый этап, а именно разбиение карты на проходимые и непроходимые поля, мы уже практически выполнили. Осталось обозначить стартовую точку.

Второй этап – основной. Мы порождаем шаги во всех направлениях (в нашем случае выбраны только ортогональные шаги, т.е. вверх-вниз, влево-вправо). При это проверяется, проходима ли соседняя точка, и не принадлежит ли она ранее обработанному множеству точек. При выполнении условий проходимости и непринадлежности её к ранее помеченным в пути ячейкам, в атрибут ячейки записывается число, равное количеству шагов от стартовой ячейки, от стартовой ячейки на первом шаге это будет 1. Каждая ячейка, меченая числом шагов от стартовой ячейки становится стартовой и из неё порождаются очередные шаги в соседние ячейки. Очевидно, что при таком переборе будет найден путь от начальной ячейки к конечной, либо очередной шаг из любой порождённой в пути ячейки будет невозможен.

Восстановление кратчайшего пути происходит в обратном направлении: при выборе ячейки от финишной ячейки к стартовой на каждом шаге выбирается ячейка, имеющая атрибут расстояния от стартовой на единицу меньше текущей ячейки. Очевидно, что таким образом находится кратчайший путь между парой заданных ячеек. Трасс с минимальной числовой длиной пути может существовать несколько.

Алгоритм:

// Инициализация

Пометить стартовую ячейку 0

d := 0

// Распространение волны

ЦИКЛ

ДЛЯ каждой ячейки loc, помеченной числом d

пометить все соседние свободные непомеченные ячейки числом d + 1

КЦ

d := d + 1

ПОКА (финишная ячейка не помечена) И (есть возможность распространения волны)

// восстановление пути

ЕСЛИ финишная ячейка помечена

ТО

перейти в финишную ячейку

ЦИКЛ

выбрать среди соседних ячейку, помеченную числом на 1 меньше числа в текущей ячейке

перейти в выбранную ячейку и добавить её к пути

ПОКА текущая ячейка — не стартовая

ВОЗВРАТ путь найден

ИНАЧЕ

ВОЗВРАТ путь не найден

## Построение минимального остовного дерева

Минимальное остовное дерево в связанном взвешенном неориентированном графе — это остовное дерево этого графа, имеющее минимальный возможный вес, где под весом дерева понимается сумма весов входящих в него рёбер.

Для построения МОД был выбран алгоритм Прима.

### Алгоритм Прима

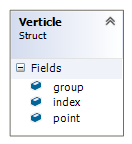
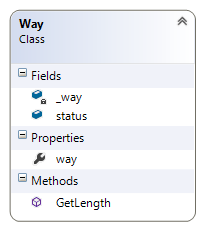
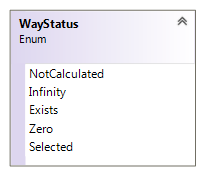
Построение начинается с дерева, включающего в себя одну (произвольную) вершину. В течение работы алгоритма дерево разрастается, пока не охватит все вершины исходного графа. На каждом шаге алгоритма к текущему дереву присоединяется самое лёгкое из рёбер, соединяющих вершину из построенного дерева и вершину не из дерева.

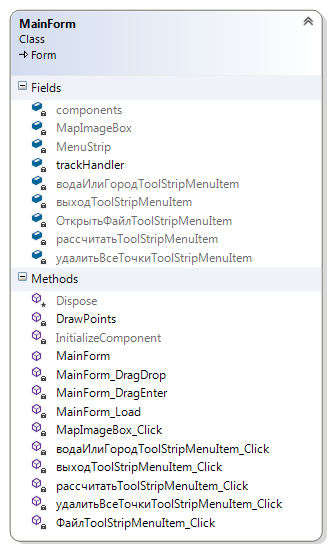
Алгоритм:

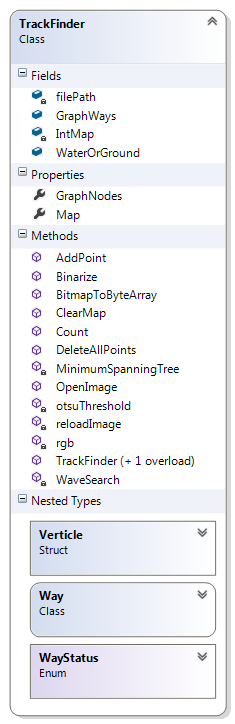
Разобъем все вершины графа на два непересекающихся множества: U и V-U. U будет служить для обработанных вершин, V-U – оставшихся. Соответственно:

1. Добавить в U произвольную вершину из V
2. Цикл: пока :
   1. Найти ребро с минимальным весом между вершинами
   2. Добавить v к U, убрать v из V-U

# Диаграммы классов





# Описание основных методов

## MainForm

|  |  |
| --- | --- |
| private void ФайлToolStripMenuItem\_Click(object sender, EventArgs e) | Обработка нажатия на кнопку «Открыть файл». Появляется диалоговое окно для поиска в проводнике файла одного из расширений (bmp, jpg, png) |
| private void DrawPoints() | Отрисовка отмеченных точек на карте. |
| private void MapImageBox\_Click(object sender, EventArgs e) | Обработка нажатий на загруженное изображение. При клике на проходимую область на карту добавляется точка. |
| private void MainForm\_DragDrop(object sender, DragEventArgs e) | Загрузка изображения при перетаскивании его в окно программы. |
| private void рассчитатьToolStripMenuItem\_Click(object sender, EventArgs e) | Обработка нажатия на кнопку «Рассчитать». При наличии точек на карте форма становится неактивной, управление передается объекту trackHandler класса TrackFinder, выполняющий расчеты. |
| private void удалитьВсеТочкиToolStripMenuItem\_Click(object sender, EventArgs e) | Удаление всех точек трассы и очистка карты от путей. |
| private void водаИлиГородToolStripMenuItem\_Click(object sender, EventArgs e) | Переключение режима загрузки карты. В скобках соответствующей кнопки указан режим, в котором будет загружаться следующее изображение. |

## TrackFinder

|  |  |
| --- | --- |
| public TrackFinder() | Пустой конструктор для первоначальной инициализации объекта-обработчика путей. |
| public TrackFinder(string filename) | Конструктор с загрузкой изображения. |
| public bool OpenImage(string filename) | Сохраняет путь к изображению filename в filePath, открывает изображение, бинаризует его, удаляет все точки с помощью соответствующих методов. Возвращает истину при удачной загрузке. |
| private void reloadImage() | Загрузка изображения из файла, путь к которому указан в filePath. |
| public void ClearMap() | Очищает карту от точек и нарисованных путей. |
| public bool AddPoint(Point p) | Пытается добавить точку в массив GraphNodes. Операция происходит удачно, если указанная точка проходима и ещё не присутствует в массиве. Также расширяется массив GraphWays: добавляется строчка и столбец. |
| public void DeleteAllPoints() | Очищает массивы точек и путей. |
| private void MinimumSpanningTree(int group) | Вычисляет минимальное остовное дерево, используя вершины внутри одной группы и их ребра в GraphNodes и GraphWays. Ребра, оставшиеся в графе, помечаются как WayStatus.Selected. |
| private Point[] WaveSearch(Point start, Point goal) | Возвращает массив точек, который означает кратчайший путь из start в goal. |
| public void Count() | Основной метод, выполняющий все операции вычисления. |
| public unsafe void Binarize(bool saveBinImage = false) | /// <summary>  /// Бинаризация изображения с сохранением каждой точки карты в массив IntMap следующим образом:  /// 0 - свободная точка;  /// -1 - занятая (пройти нельзя).  /// </summary>  /// <param name="saveBinImage">Указать истину, если надо сохранить бинаризированное изображение  /// в файл "new.bmp", по умолчанию файл не сохраняется.</param> |
| public unsafe byte[,,] BitmapToByteArray()  private int otsuThreshold()  private int rgb(byte[, ,] mass, int j, int i) | Вспомогательные методы для метода Binarize (преобразование изображения в массив RGB точек, вычисление порога Оцу, метод для проверки цвета в методе Оцу) |

# Листинг программы

## Tracks - Main.cs

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Drawing.Imaging;

using System.IO;

using System.Linq;

using System.Text.RegularExpressions;

using System.Threading.Tasks;

namespace Tracks

{

public class TrackFinder

{

// 0 - вода, 1 - земля

public bool WaterOrGround;

// изображение для вывода и обработки

public Bitmap Map { get; private set; }

// сохраняем путь для перезагрузки изображения

private string filePath;

// 0 (-2) - свободно, -1 (-1) - занято

private int[,] IntMap;

// конструкторы

public TrackFinder()

{

}

public TrackFinder(string filename)

{

OpenImage(filename);

}

// загрузка изображения

public bool OpenImage(string filename)

{

filePath = filename;

try

{

reloadImage();

}

catch (Exception)

{

return false;

}

Binarize(true);

DeleteAllPoints();

return true;

}

private void reloadImage()

{

var fs = new FileStream(filePath, FileMode.**Open**);

try

{

Map = new Bitmap(fs);

}

catch (Exception)

{

throw;

}

finally

{

fs.Close();

}

}

// очищаем карту от нарисованных путей через замену её на оригинал

public void ClearMap()

{

DeleteAllPoints();

if (Map == null)

return;

using (var fs = new FileStream(filePath, FileMode.**Open**)) {

Map = new Bitmap(fs);

}

}

// описание графа

public class MyPoint

{

public Point p;

//private int group;

public int Group { get; set; }

public MyPoint(Point p)

{

this.p = p;

}

}

// координаты точек

public List<MyPoint> GraphNodes { get; private set; }

// добавление/удаление точек на карте

public bool AddPoint(Point p)

{

if (IntMap[p.X, p.Y] == -1) return false;

GraphNodes.Add(new MyPoint(p));

// добавляем точку с расширением матрицы графа

int a = GraphWays == null ? 1 : GraphWays.GetLength(0) + 1;

int b = GraphWays == null ? 1 : GraphWays.GetLength(1) + 1;

var newGraphWays = new Way[a, b];

for (int **i** = 0; **i** < a - 1; **i**++)

for (int **j** = 0; **j** < b - 1; **j**++)

newGraphWays[**i**, **j**] = GraphWays[**i**, **j**];

for (int **i** = 0; **i** < newGraphWays.GetLength(0); **i**++)

newGraphWays[**i**, newGraphWays.GetLength(1) - 1] = new Way();

for (int **j** = 0; **j** < newGraphWays.GetLength(1); **j**++)

newGraphWays[newGraphWays.GetLength(0) - 1, **j**] = new Way();

newGraphWays[newGraphWays.GetLength(0) - 1, newGraphWays.GetLength(0) - 1].status = WayStatus.**Zero**;

GraphWays = newGraphWays;

return true;

}

public void DeleteAllPoints()

{

GraphNodes = new List<MyPoint>();

GraphWays = null;

}

public enum WayStatus { **NotCalculated**, **Infinity**, **Exists**, **Zero**, **Selected** }

public class Way

{

public WayStatus status = WayStatus.**NotCalculated**;

private Point[] \_way;

public Point[] way

{

get { return \_way; }

set

{

if (value == null)

status = WayStatus.**Infinity**;

else

{

status = WayStatus.**Exists**;

\_way = value;

}

}

}

public int GetLength()

{

if (status == WayStatus.**Infinity**)

return Int32.**MaxValue**;

if (status == WayStatus.**Zero** || status == WayStatus.**NotCalculated**)

return 0;

return \_way.GetLength(0);

}

}

// ребра графа

public Way[,] GraphWays;

// поиск минимального остовного дерева алгоритмом Прима

private void MinimumSpanningTree(int group)

{

var VminusU = new List<Verticle>(GraphNodes.Count);

VminusU.AddRange(GraphNodes.Select((t, i) => new Verticle {index = i, point = t.p, group = t.Group}).

Where(gn => gn.group == group));

if (!VminusU.Any())

return;

var U = new List<Verticle>(GraphNodes.Count)

{

new Verticle {index = VminusU[0].index, point = VminusU[0].point}

};

VminusU.RemoveAt(0);

while (VminusU.Any())

{

var **minWay** = new Way();

int minIGW = 0, minJGW = 0, minJ = 0;

for (var **i** = 0; **i** < U.Count(); **i**++)

for (var **j** = 0; **j** < VminusU.Count(); **j**++)

if (GraphWays[U[**i**].index, VminusU[**j**].index].status == WayStatus.**Exists** &&

(**minWay**.status == WayStatus.**NotCalculated** ||

GraphWays[U[**i**].index, VminusU[**j**].index].GetLength() < **minWay**.GetLength()))

{

minIGW = U[**i**].index;

minJGW = VminusU[**j**].index;

minJ = **j**;

**minWay** = GraphWays[U[**i**].index, VminusU[**j**].index];

}

GraphWays[minIGW, minJGW].status = WayStatus.**Selected**;

U.Add(new Verticle {index = VminusU[minJ].index, point = VminusU[minJ].point});

VminusU.RemoveAt(minJ);

}

}

// вспомогательная структура

private struct Verticle

{

public Point point;

public int index; // индекс вершины в GraphNodes

public int group;

}

// волновой поиск

private Point[] WaveSearch(Point start, Point goal)

{

// соседи

Point[] neighBoughrs = {new Point(0,-1), new Point(0,1), new Point(1,0), new Point(-1,0)};

// копируем карту для внесения изменений

var tempMap = new int[IntMap.GetLength(0), IntMap.GetLength(1)];

for (int **i** = 0; **i** < IntMap.GetLength(0); **i**++)

{

for (int **j** = 0; **j** < IntMap.GetLength(1); **j**++)

{

if (IntMap[**i**, **j**] == 0)

tempMap[**i**, **j**] = -2;

else

tempMap[**i**, **j**] = -1;

}

}

// изначально волна - 0. отмечаем ею стартовую точку

int **wave**;

tempMap[start.X, start.Y] = **wave** = 0;

while (true)

{

bool enough = true;

for (int **i** = 0; **i** < tempMap.GetLength(0); **i**++)

for (int **j** = 0; **j** < tempMap.GetLength(1); **j**++)

if (tempMap[**i**, **j**] == **wave**)

for (int **k** = 0; **k** < 4; **k**++)

if (**i** + neighBoughrs[**k**].X >= 0 && **i** + neighBoughrs[**k**].X < tempMap.GetLength(0) &&

**j** + neighBoughrs[**k**].Y >= 0 && **j** + neighBoughrs[**k**].Y < tempMap.GetLength(1))

if (tempMap[**i** + neighBoughrs[**k**].X, **j** + neighBoughrs[**k**].Y] == -2)

{

enough = false;

tempMap[**i** + neighBoughrs[**k**].X, **j** + neighBoughrs[**k**].Y] = **wave** + 1;

}

**wave**++;

if (enough)

break;

if (tempMap[goal.X, goal.Y] != -2)

break;

}

// если путь не найден (его нет)

if (tempMap[goal.X, goal.Y] == -2)

return null;

// восстановить путь

int length = tempMap[goal.X, goal.Y];

var lp = new List<Point>();

var **temp** = new Point(goal.X, goal.Y);

lp.Add(**temp**);

for (var **i** = length - 1; **i** > 0; **i**--)

{

for (int **k** = 0; **k** < 4; **k**++)

if (**temp**.X + neighBoughrs[**k**].X >= 0 && **temp**.X + neighBoughrs[**k**].X < tempMap.GetLength(0) &&

**temp**.Y + neighBoughrs[**k**].Y >= 0 && **temp**.Y + neighBoughrs[**k**].Y < tempMap.GetLength(1))

if (tempMap[**temp**.X + neighBoughrs[**k**].X, **temp**.Y + neighBoughrs[**k**].Y] == **i**)

{

**temp** = new Point(**temp**.X + neighBoughrs[**k**].X, **temp**.Y + neighBoughrs[**k**].Y);

lp.Add(**temp**);

break;

}

}

lp.Add(start);

return lp.ToArray();

}

// основной метод класса

public void ToCount()

{

if (Map == null)

return;

reloadImage();

if (GraphWays == null)

return;

for (int **i** = 0; **i** < GraphNodes.Count; **i**++)

{

GraphNodes[**i**].Group = **i**;

for (int **j** = 0; **j** < GraphNodes.Count; **j**++)

{

if (GraphWays[**i**, **j**].status == WayStatus.**Selected**)

GraphWays[**i**, **j**].status = WayStatus.**Exists**;

}

}

var po = new ParallelOptions {MaxDegreeOfParallelism = Environment.ProcessorCount - 1};

Parallel.For(0, GraphNodes.Count - 1, po, i => Parallel.For(i + 1, GraphNodes.Count, po, j =>

{

if (GraphWays[i, j].status == WayStatus.**NotCalculated**)

if (GraphNodes[i].p == GraphNodes[j].p)

{

GraphWays[i, j].status = WayStatus.**Zero**;

GraphWays[j, i].status = WayStatus.**Zero**;

}

else

{

GraphWays[i, j].way = WaveSearch(GraphNodes[i].p, GraphNodes[j].p);

GraphWays[j, i].way = GraphWays[i, j].way;

if (GraphWays[i, j].way != null)

GraphWays[i, j].status = GraphWays[j, i].status = WayStatus.**Exists**;

else

GraphWays[i, j].status = GraphWays[j, i].status = WayStatus.**Infinity**;

}

}));

// расставляем группы связности

for (int **i** = 0; **i** < GraphNodes.Count - 1; **i**++)

for (int **j** = **i** + 1; **j** < GraphNodes.Count; **j**++)

{

if (GraphWays[**i**, **j**].status == WayStatus.**Exists**)

GraphNodes[**j**].Group = GraphNodes[**i**].Group;

}

for (int **i** = 0; **i** <= GraphNodes.OrderByDescending(gn => gn.Group).First().Group; **i**++)

MinimumSpanningTree(**i**);

foreach (var **graphWay** in GraphWays)

{

if (**graphWay**.status == WayStatus.**Selected** && **graphWay**.way != null)

{

try

{

foreach (var **point** in **graphWay**.way)

Map.SetPixel(**point**.X, **point**.Y, Color.Red);

}

catch (Exception)

{

}

}

}

}

/// <summary>

/// Бинаризация изображения с сохранением каждой точки карты в массив IntMap следующим образом:

/// 0 - свободная точка;

/// -1 - занятая (пройти нельзя).

/// </summary>

/// <param name="saveBinImage">Указать истину, если надо сохранить бинаризированное изображение

/// в файл "new.bmp", по умолчанию файл не сохраняется.</param>

public unsafe void Binarize(bool saveBinImage = false)

{

int otsu = otsuThreshold();

int width = Map.Width;

int height = Map.Height;

var ret = new int[width, height];

BitmapData bd = Map.LockBits(new Rectangle(0, 0, width, height), ImageLockMode.**ReadOnly**,

PixelFormat.**Format24bppRgb**);

Bitmap output = new Bitmap(width, height, PixelFormat.**Format24bppRgb**);

BitmapData outputbd = output.LockBits(new Rectangle(0, 0, width, height), ImageLockMode.**WriteOnly**,

PixelFormat.**Format24bppRgb**);

try

{

byte\* **curpos**;

byte\* **curposnew**;

for (int **h** = 0; **h** < height; **h**++)

{

**curpos** = ((byte\*)bd.Scan0) + **h** \* bd.Stride;

**curposnew** = ((byte\*)outputbd.Scan0) + **h** \* bd.Stride;

for (int **w** = 0; **w** < width; **w**++)

{

int b = \*(**curpos**++);

int g = \*(**curpos**++);

int r = \*(**curpos**++);

if ((r + b + g) / 3 < otsu && WaterOrGround || !WaterOrGround && r\*r + g\*g < b\*b)

{

\*(**curposnew**++) = 0;

\*(**curposnew**++) = 0;

\*(**curposnew**++) = 0;

ret[**w**, **h**] = -1;

}

else

{

\*(**curposnew**++) = 255;

\*(**curposnew**++) = 255;

\*(**curposnew**++) = 255;

ret[**w**, **h**] = 0;

}

}

}

}

finally

{

Map.UnlockBits(bd);

output.UnlockBits(outputbd);

}

if (saveBinImage)

output.Save(@"d:\new.bmp");

IntMap = ret;

}

// вспомогательные методы

public unsafe byte[,,] BitmapToByteArray()

{

var binaryMapByteArray = new byte[3, Map.Height, Map.Width];

BitmapData bd = Map.LockBits(new Rectangle(0, 0, Map.Width, Map.Height), ImageLockMode.**ReadOnly**,

PixelFormat.**Format24bppRgb**);

try

{

for (int **h** = 0; **h** < Map.Height; **h**++)

{

byte\* **curpos** = ((byte\*)bd.Scan0) + **h** \* bd.Stride;

for (int **w** = 0; **w** < Map.Width; **w**++)

{

binaryMapByteArray[2, **h**, **w**] = \*(**curpos**++);

binaryMapByteArray[1, **h**, **w**] = \*(**curpos**++);

binaryMapByteArray[0, **h**, **w**] = \*(**curpos**++);

}

}

}

finally

{

Map.UnlockBits(bd);

}

return binaryMapByteArray;

}

private int otsuThreshold()

{

byte[,,] binaryMapByteArray = BitmapToByteArray();

int **min** = rgb(binaryMapByteArray, 0, 0), **max** = rgb(binaryMapByteArray, 0, 0);

int **temp**, **temp1**;

/\*\*\*\* Построение гистограммы \*\*\*\*/

/\* Узнаем наибольший и наименьший полутон \*/

for (int **i** = 0; **i** < Map.Width; **i**++)

{

for (int **j** = 0; **j** < Map.Height; **j**++)

{

**temp** = rgb(binaryMapByteArray, **i**, **j**);

if (**temp** < **min**) **min** = **temp**;

if (**temp** > **max**) **max** = **temp**;

}

}

int[] hist = new int[**max** - **min** + 1];

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

hist[**i**] = 0;

/\* Считаем сколько каких полутонов \*/

for (int **i** = 0; **i** < Map.Width; **i**++)

for (int **j** = 0; **j** < Map.Height; **j**++)

hist[rgb(binaryMapByteArray, **i**, **j**) - **min**]++;

// Гистограмма построена

int **alpha**, **beta**, threshold = 0;

**temp** = **temp1** = 0;

**alpha** = **beta** = 0;

/\* Для расчета математического ожидания первого класса \*/

for (int **i** = 0; **i** <= (**max** - **min**); **i**++)

{

**temp** += **i** \* hist[**i**];

**temp1** += hist[**i**];

}

double **sigma**, **maxSigma** = -1;

double **w1**, **a**;

/\* Основной цикл поиска порога

Пробегаемся по всем полутонам для поиска такого, при котором внутриклассовая дисперсия минимальна \*/

for (int **i** = 0; **i** < (**max** - **min**); **i**++)

{

**alpha** += **i** \* hist[**i**];

**beta** += hist[**i**];

**w1** = (double)**beta** / **temp1**;

**a** = (double)**alpha** / **beta** - (double)(**temp** - **alpha**) / (**temp1** - **beta**);

**sigma** = **w1** \* (1 - **w1**) \* **a** \* **a**;

if (**sigma** > **maxSigma**)

{

**maxSigma** = **sigma**;

threshold = **i**;

}

}

return threshold + **min**;

}

private int rgb(byte[, ,] mass, int j, int i)

{

return (mass[0, i, j] + mass[1, i, j] + mass[2, i, j]) / 3;

}

}

}

## GUI – Program.cs

using System;

using System.Windows.Forms;

namespace GUI

{

static class Program

{

/// <summary>

/// The main entry point for the application.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new MainForm());

}

}

}

## GUI – MainForm.cs

using System;

using System.Drawing;

using System.Linq;

using System.Windows.Forms;

using Tracks;

namespace GUI

{

public partial class MainForm : Form

{

private TrackFinder trackHandler;

public MainForm()

{

InitializeComponent();

}

private void MainForm\_Load(object sender, EventArgs e)

{

AllowDrop = true;

trackHandler = new TrackFinder();

водаИлиГородToolStripMenuItem.Text = @"Смена типа местности (земля)";

trackHandler.WaterOrGround = true;

//trackHandler.OpenImage(@"d:\Capture3.png");

//MapImageBox.Image = trackHandler.Map;

}

private void выходToolStripMenuItem\_Click(object sender, EventArgs e)

{

Application.Exit();

}

private void ФайлToolStripMenuItem\_Click(object sender, EventArgs e)

{

var openFileDialog1 = new OpenFileDialog

{

Filter = @"Файл изображения|\*.bmp;\*.jpg;\*.png",

Title = @"Выберите изображение"

};

if (openFileDialog1.ShowDialog() != DialogResult.**OK**) return;

trackHandler.OpenImage(openFileDialog1.FileName);

MapImageBox.Image = trackHandler.Map;

}

private void DrawPoints()

{

var gr = MapImageBox.CreateGraphics();

if (trackHandler.GraphNodes == null || !trackHandler.GraphNodes.Any())

return;

foreach (var **graphNode** in trackHandler.GraphNodes)

{

gr.FillEllipse(Brushes.Red, new Rectangle(**graphNode**.p.X - 5,

**graphNode**.p.Y - 5, 10, 10));

}

}

private void MapImageBox\_Click(object sender, EventArgs e)

{

// если картинки нету, ничего не делать

if (trackHandler.Map == null)

return;

Point click = new Point(MapImageBox.PointToClient(Cursor.Position).Y, MapImageBox.PointToClient(Cursor.Position).X);

// если точка поставлена вне изображения

if (click.Y >= trackHandler.Map.Width || click.X >= trackHandler.Map.Height ||

click.Y < 0 || click.X < 0)

return;

if (trackHandler.AddPoint(new Point(click.Y, click.X)))

DrawPoints();

}

private void MainForm\_DragEnter(object sender, DragEventArgs e)

{

e.Effect = e.Data.GetDataPresent(DataFormats.FileDrop) ? DragDropEffects.**Copy** : DragDropEffects.**None**;

}

private void MainForm\_DragDrop(object sender, DragEventArgs e)

{

e.Effect = DragDropEffects.**None**;

if (!e.Data.GetDataPresent(DataFormats.FileDrop))

return;

if (!(trackHandler.OpenImage(((string[]) e.Data.GetData(DataFormats.FileDrop))[0])))

{

MapImageBox.Image = MapImageBox.ErrorImage;

return;

}

MapImageBox.Image = trackHandler.Map;

trackHandler.DeleteAllPoints();

}

private void рассчитатьToolStripMenuItem\_Click(object sender, EventArgs e)

{

this.Enabled = false;

trackHandler.ToCount();

this.Enabled = true;

MapImageBox.Image = trackHandler.Map;

DrawPoints();

}

private void удалитьВсеТочкиToolStripMenuItem\_Click(object sender, EventArgs e)

{

trackHandler.ClearMap();

MapImageBox.Image = trackHandler.Map;

}

private void водаИлиГородToolStripMenuItem\_Click(object sender, EventArgs e)

{

trackHandler.WaterOrGround = !trackHandler.WaterOrGround;

водаИлиГородToolStripMenuItem.Text = trackHandler.WaterOrGround ?

"Смена типа местности (земля)" : "Смена типа местности (вода)";

}

}

}

# Список литературы

Рихтер Дж. - CLR via C#. Программирование на платформе Microsoft .NET Framework 4.0 на языке C# (Мастер-класс) – 2012

<http://msdn.microsoft.com/>

[Метод Оцу](http://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%9E%D1%86%D1%83)

[Алгоритм Прима](http://algolist.manual.ru/maths/graphs/span.php)

[Волновой поиск](http://www.codenet.ru/progr/alg/way.php)

[Бинаризация изображения](http://habrahabr.ru/post/196578/)