***Министерство образования Республики Беларусь***

***Учреждение образования***

***«Брестский государственный технический университет»***

***Кафедра ИИТ***

**Лабораторная работа №7**

**По дисциплине МОИС за III семестр**

**Тема: «Построение выпуклой оболочки»**

**Выполнил:**

Студент группы ИИ-15 (1)

2-го курса

Волк И. А.

**Проверил:**

Шуть В. Н.

Брест 2018

Цель: научиться строить выпуклую оболочку.

***Задание.***

1. Построить выпуклую оболочку, используя обход по Джарвису или сканирование по Грэхему. Вход: точки исходного множества, заданные координатами. Выход: вершины выпуклой оболочки.
2. \*Реализовать визуализацию построения через консольное или оконное приложение.
3. Построить выпуклую оболочку самостоятельно выбранным алгоритмом, имеющим меньшее время работы по сравнению с алгоритмом Джарвиса. Обосновать выбор метода.
4. Обосновать, в чем преимущество использования векторного произведения по сравнению с явным вычислением углов.

Код программы:

**//main.cpp**

#include <iostream>

#include <set>

#include <chrono>

#include "lab7\_1.h"

#define out(a) {\

int i = 0;\

for (Point p : a)\

{\

std::cout << ((i) ? ", " : "") << "(" << p.first << ";" << p.second << ")";\

i++;\

}\

std::cout << std::endl; }

int main()

{

using namespace Lab7;

PointSet points = { {7.0, 1.0},

{12.0, 2.0},

{9.0, 3.0},

{11.0, 4.0},

{7.0, 5.0},

{6.0, 5.0},

{13.0, 5.0},

{16.0, 7.0},

{13.0, 8.0},

{11.0, 7.0},

{9.0, 6.0},

{5.0, 7.0},

{3.0, 7.0},

{8.0, 9.0},

{9.0, 11.0},

{12.0, 10.0},

{14.0, 11.0} };

PointSet convexHull1 = JarvisMarch(points);

out(convexHull1);

PointSet convexHull2 = GrahamScan(points);

out(convexHull2);

PointSet convexHull3 = Chan(points, 3);

out(convexHull3);

std::cout << std::endl;

std::cin.get();

}

**//lab7\_1.h**

#pragma once

#include <iostream>

#include <vector>

namespace Lab7

{

typedef std::pair<double, double> Point;

typedef std::vector<Point> PointSet;

PointSet JarvisMarch(const PointSet& initSet);

PointSet Chan(PointSet& initSet, int m);

PointSet GrahamScan(PointSet& initSet);

}; //Lab7

**//lab7\_1.cpp**

#include "stdafx.h"

#include "lab7\_1.h"

#include <stack>

#include <vector>

#include <algorithm>

# define M\_PI 3.14159265358979323846

namespace Lab7

{

static PointSet& operator+=(PointSet& set, const PointSet& other)

{

for (Point p : other)

set.push\_back(p);

return set;

}

static double **angle**(const Point& d1, const Point& d2, const Point& d0)

{

return (d1.first - d0.first)\*(d2.second - d0.second) –

(d1.second - d0.second)\*(d2.first - d0.first);

}

static double **length**(const Point& d1, const Point& d2)

{

return pow(pow(d2.first - d1.first,2.0) + pow(d2.second - d1.second,2.0), 0.5);

}

PointSet **JarvisMarch**(const PointSet& initSet)

{

if (initSet.empty())

return PointSet();

Point down = initSet[0];

for (Point p : initSet)

if (p.second < down.second)

down = p;

else if (p.second == down.second)

{

if (p.first < down.first)

down = p;

}

PointSet result;

result.push\_back(down);

Point current = down;

while (true)

{

Point next;

for (int i = 0; i < initSet.size(); i++) // select any

if (initSet[i] == current)

continue;

else

{

next = initSet[i];

break;

}

for (int i = 0; i < initSet.size(); i++)

{

if (initSet[i] == current || initSet[i] == next)

continue;

if (angle(next, initSet[i], current) < 0)

{

next = initSet[i];

}

else if (angle(next, initSet[i], current) == 0)

{

if (length(current, next) < length(current, initSet[i]))

next = initSet[i];

}

}

if (next == down)

break;

result.push\_back(next);

current = next;

}

return result;

}

PointSet **GrahamScan**(PointSet& initSet)

{

if (initSet.empty())

return PointSet();

Point down = initSet[0]; // lowest

for (Point p : initSet)

if (p.second < down.second)

down = p;

else if (p.second == down.second)

{

if (p.first < down.first)

down = p;

}

std::sort(initSet.begin(), initSet.end(),

[&](Point& f, Point& s)

{

if (f == down)

return true;

if (s == down)

return false;

double ang = angle(f, s, down);

if (ang > 0)

return true;

else if (ang == 0)

if (length(down, f) >= length(down, s))

return true;

else

return false;

return false;

});

PointSet result;

result.push\_back(initSet[0]);

result.push\_back(initSet[1]);

for (int i = 2; i < initSet.size(); i++)

{

while (result.size() > 1

&& angle(result.back(), initSet[i], \*(result.end() - 2)) <= 0)

result.pop\_back();

result.push\_back(initSet[i]);

}

return result;

}

static PointSet **GrahamScan**(PointSet& initSet, int begin, int end)

{

if (initSet.empty())

return PointSet();

Point down = initSet[begin]; // lowest

for (int i = begin; i < end; i++)

if (initSet[i].second < down.second)

down = initSet[i];

else if (initSet[i].second == down.second)

{

if (initSet[i].first < down.first)

down = initSet[i];

}

PointSet result;

result.push\_back(initSet[begin]);

result.push\_back(initSet[begin+1]);

for (int i = begin + 2; i < end; i++)

{

while (result.size() > 1 &&

angle(result.back(), initSet[i], \*(result.end() - 2)) <= 0)

result.pop\_back();

result.push\_back(initSet[i]);

}

return result;

}

PointSet **Chan**(PointSet& initSet, int m)

{

Point down = initSet[0]; // lowest

for (Point p : initSet)

if (p.second < down.second)

down = p;

else if (p.second == down.second)

{

if (p.first < down.first)

down = p;

}

std::sort(initSet.begin(), initSet.end(),

[&](Point& f, Point& s)

{

if (f == down)

return true;

if (s == down)

return false;

double ang = angle(f, s, down);

if (ang > 0)

return true;

else if (ang == 0)

if (length(down, f) >= length(down, s))

return true;

else

return false;

return false;

});

PointSet parts;

int partSize = initSet.size() / m;

int i = 0;

for (; i + 2 \* partSize < initSet.size(); i += partSize)

{

parts += GrahamScan(initSet, i, i + partSize);

}

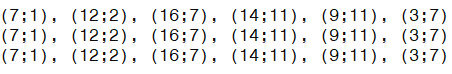
parts += GrahamScan(initSet, i, initSet.size());

return JarvisMarch(parts);

}

}; //Lab7

Результат выполнения:



Вывод: научился кодировать и раскодировать деревья.