**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ «МИСиС»**

*КАФЕДРА ИНЖЕНЕРНОЙ КИБЕРНЕТИКИ*

*НАПРАВЛЕНИЕ ПРИКЛАДНАЯ МАТЕМАТИКА ГРУППА БПМ-16-2*

### Домашнее задание

*ПО КУРСУ*: «Математические методы в компьютерной графике»

*ТЕМА*: «Выпуклая оболочка на плоскости. Алгоритм Эндрю»

*СТУДЕНТ: Лившиц Ася*

*ПРЕПОДАВАТЕЛЬ: Куренкова Т. В.*

Москва, 2019 г.

Оглавление

[Домашнее задание 1](#_Toc28601241)

[Введение 3](#_Toc28601242)

[Теоретическое введение 4](#_Toc28601243)

[Постановка задачи 5](#_Toc28601244)

[Математическая постановка задачи 6](#_Toc28601245)

[Описание алгоритма 7](#_Toc28601246)

[Средства разработки 8](#_Toc28601247)

[Технические требования: 8](#_Toc28601248)

[Список файлов, необходимы для корректной работы программы 9](#_Toc28601249)

[Взаимодействие с пользователем 10](#_Toc28601250)

[Результаты работы программы 11](#_Toc28601251)

[Приложение 15](#_Toc28601252)

[Список использованной литературы: 26](#_Toc28601253)

# Введение

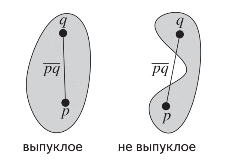
Нахождение выпуклой оболочки на плоскости является одной из классических задач Вычислительной геометрии.

В терминах вычислительной геометрии задача связана с нахождением многоугольника с минимальной площадью из максимальных, такого, что все заданные точки являются либо его вершинами, либо лежат внутри него.

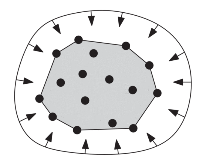
К данной задаче имеется множество разных подходов решения. Существуют статические выпуклые оболочки (алгоритмы Джарвиса, Грэхема, Эндрю, Чена, а также алгоритм быстрой оболочки), динамический алгоритм построения.

В данной работе будет рассмотрено применение алгоритма Эндрю, как одного из простейших в реализации, а также показывающего неплохую скорость работы.

# Теоретическое введение

Множество на плоскости называется выпуклым, если для любой пары точек , принадлежащих множеству () каждая точка отрезка также принадлежит множеству (). Иначе многоугольник является невыпуклым.

Будем рассматривать задачу нахождения выпуклой оболочки для конечного множества точек.

Для понимания термина «выпуклая оболочка» приведем следующую метафору. Пусть в некоторых местах плоскости вбиты колышки. Накинем достаточно большую веревочную петлю на них таким образом, чтобы все колышки заведомо помещались внутри нее. Если петлю достаточно сильно затянуть (предположим, что колышки достаточно крепкие, чтобы выдерживать большое усилие), то веревка натянется на некоторые из колышков таким образом, что все остальные окажутся внутри петли. Именно те колышки, на которые натянулась веревка, являются материальными аналогами вершин выпуклой оболочки.

Таким образом, выпуклая оболочка – однозначно определенный выпуклый многоугольник, вершинами которого являются точки из заданного множества, и содержащий все оставшиеся точки этого множества.

Под обходом выпуклого многоугольника будем понимать перестановку номеров вершин (или их координат), такую, что первой стоит вершина с минимальными координатами по осям , а дальнейшие вершины стоят в порядке, соответствующем нумерации вершин «по часовой стрелке».

# Постановка задачи

Создать программное средство, реализующее алгоритм построения выпуклой оболочки методом Эндрю. Должно быть обеспечено следующее:

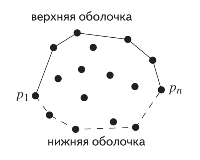
1. Возможность пользователю задавать и изменять входные данные;
2. Возможность получить визуализацию результата работы программы интуитивно понятным интерфейсом;
3. Возможность сохранить результат (координаты полученного многоугольника) в файл и просмотр этого результата;
4. Наличие краткой инструкции по работе с программным средством.

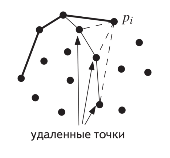
# Математическая постановка задачи

Дано множество , состоящее из точек, заданных покоординатно (каждая точка представляет пару чисел – координаты по и ). Требуется найти с помощь алгоритма Эндрю построения выпуклой оболочки такое подмножество множества (), которое:

1. Образует выпуклый многоугольник;
2. Все точки множества, которые не входят в данное подмножество (), лежат внутри или на границе многоугольника из точек .

# Описание алгоритма

Алгоритм Эндрю представляет собой модификацию алгоритма Грэхема. На каждом шаге будем рассматривать очередную вершину и пересчитывать ответ относительно нее. Будем по отдельности строить выпуклые оболочки для точек, лежащих в нижней половине (ниже отрезка, соединяющего самую левую и самую правую точки) и верхней половине.

1. Отсортируем точки по возрастанию координаты , а при равенстве – по возрастанию координаты . Получили последовательность точек, к каждому элементу которой мы будем последовательно применять следующие шаги;
2. Используя факт, что при обходе выпуклого многоугольника по часовой стрелке происходит поворот «направо», будем откатывать изменения в том случае, если при проверке очередной точки с предыдущими двумя образуется поворот «налево». В этом случае необходимо откатить все предыдущие изменения (удалять среднюю точку в тройке) выпуклой оболочки до тех пор, пока поворот не станет «правым». При этом, проверка и изменения осуществляются отдельно для верхней и нижней половин.

Асимптотика (примерное время работы по числу операций) – при памяти. Алгоритм Эндрю работает быстрее алгоритма Грэхема за счет разделения на 2 половины, благодаря чему тратится меньшее число шагов.

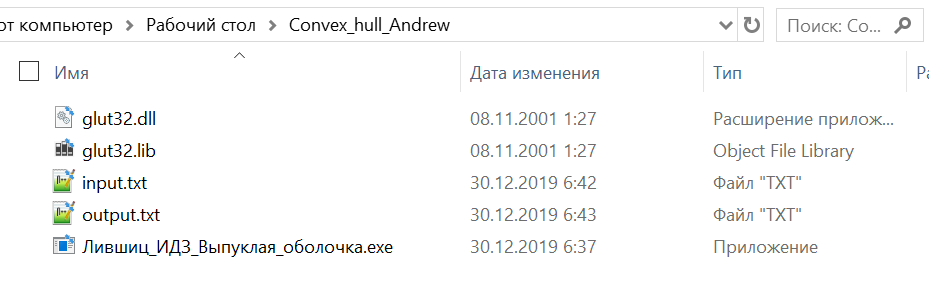
# Средства разработки

* Язык C++ 11;
* Программная среда разработки – Visual Studio 2017;
* Сторонние библиотеки – OpenGL.

# Технические требования:

* Операционная система – Windows 8 и выше;
* Процессор с тактовой частотой 1.6 ГГц или выше.

# Список файлов, необходимы для корректной работы программы



Для корректного запуска в папке находится файл “glut.h”, который перед запуском необходимо поместить по следующему пути:

**C:\Program Files (x86)\Windows Kits\10\Include\10.0.14393.0\um\gl**

Для запуска необходимо выполнить файл “Лившиц\_ИДЗ\_Выпуклая\_оболочка.exe”. Предварительно необходимо ознакомиться с форматом ввода в файл “input.txt” (см. след. Стр.) и ввести данные.

# Взаимодействие с пользователем

Обмен данными пользователя с программой осуществляется через файлы ввода или вывода, а также окно визуализации.

Файл ввода данных “input.txt” поддерживает следующий формат входных данных:

* Первая строка – одно число – число точек для построения выпуклой оболочки;
* Вторая и последующие строк – пары координат , разделенные пробелом.

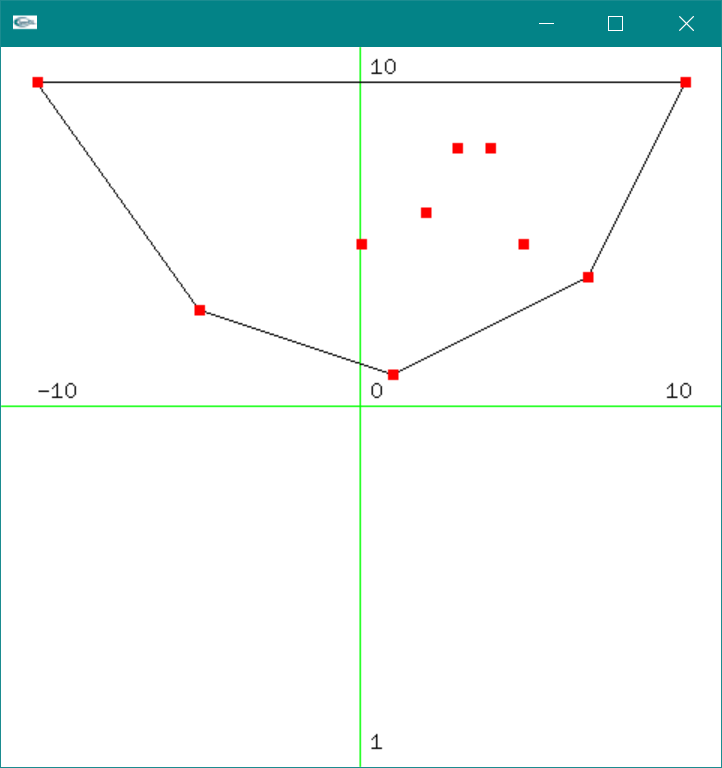
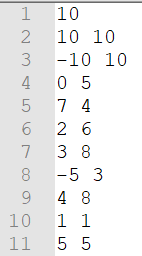
Файл, содержащий результат работы программы – “output.txt” поддерживает следующий формат вывода данных:

* Число точек, составляющих выпуклую оболочку;
* Координаты точек выпуклой оболочки начиная с левой нижней, в порядке обхода по часовой стрелке;
* Комментарии для обозначения вида выходных данных.

**Важно:**

Имена данных файлов менять не рекомендуется, так как это негативно повлияет на работу программы.

# Результаты работы программы



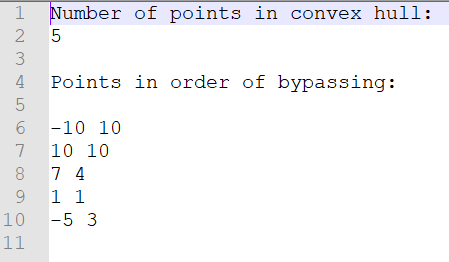
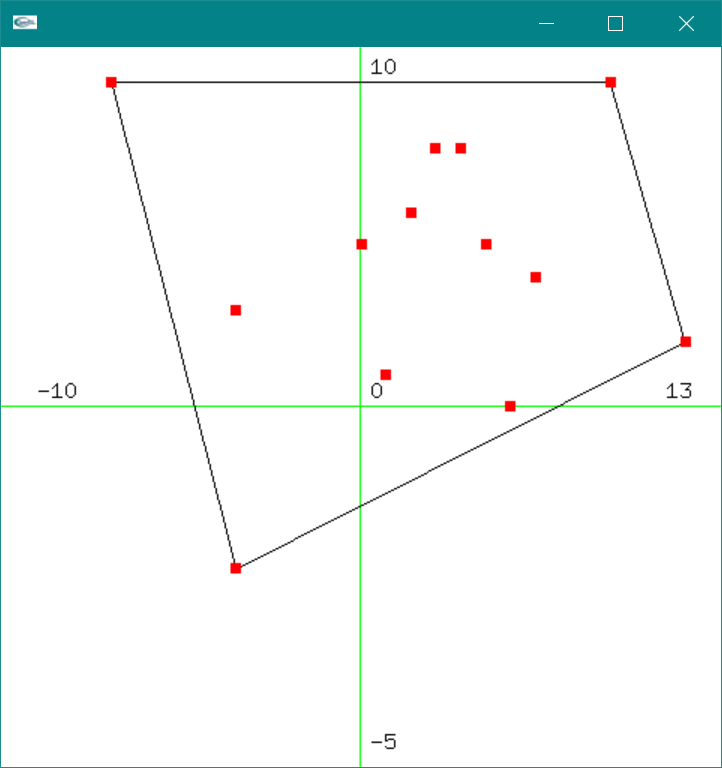
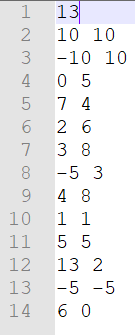


Рис. 1





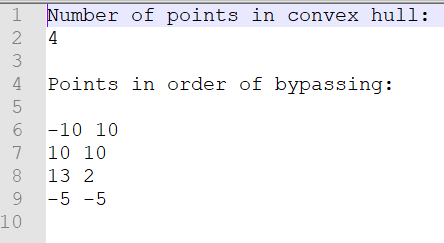
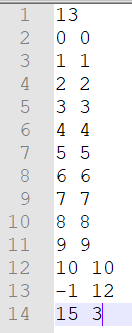
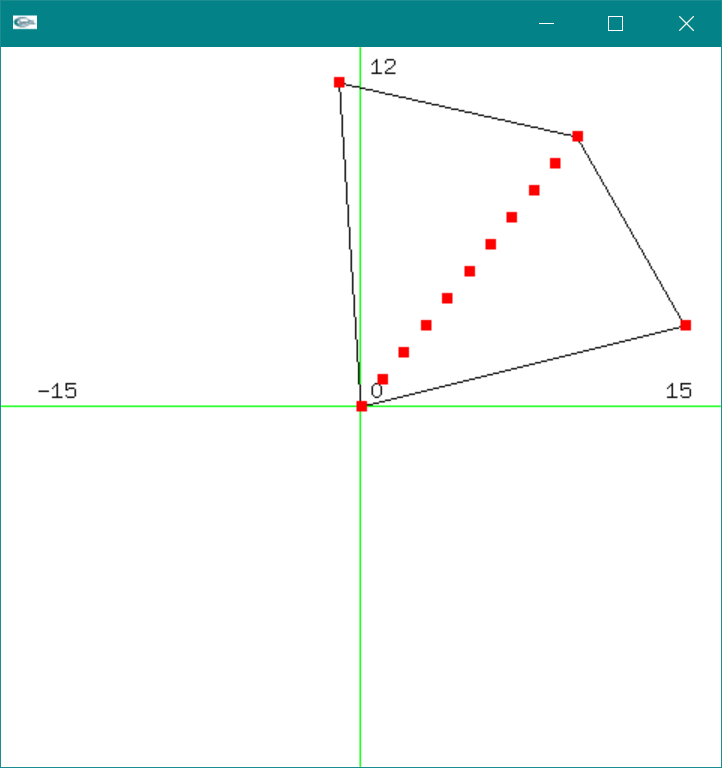


Рис. 2



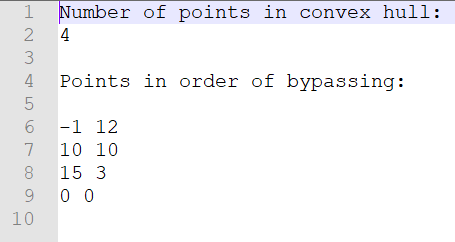
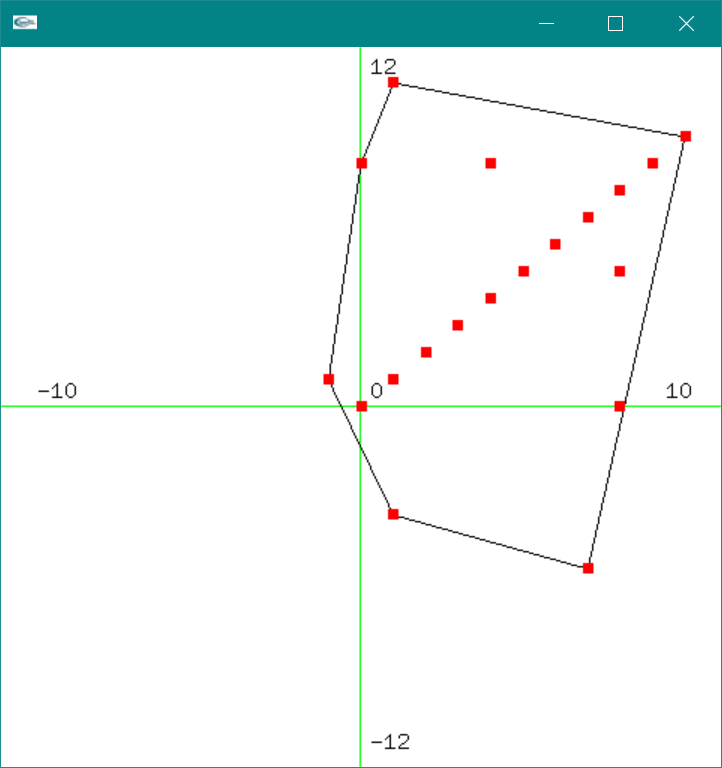
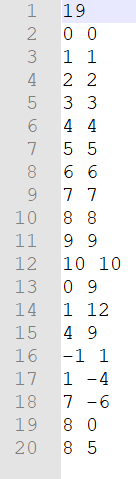


Рис. 3



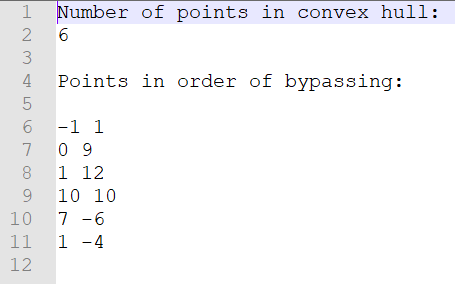


Рис. 4

# Приложение

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <vector>

#include <set>

#include <map>

#include <queue>

#include <bitset>

#include <cmath>

#include <sstream>

#include <string>

#include <algorithm>

#include <tuple>

#include <iomanip>

#include <stack>

#include <gl/glut.h>

#define endl "\n"

#define ft first

#define sd second

#define mp make\_pair

#define mt make\_tuple

#define pb push\_back

#define deb(x) for(auto temp : (x))cout<<temp<<" ";cout<<endl;

#define debs(x) cout<<(x).size()<<": ";for(auto temp : (x))cout<<temp<<" ";cout<<endl;

#define cin\_ ios\_base::sync\_with\_stdio(0);cin.tie(0);

#define cin\_pres cout<<fixed<<setprecision(10);

#define all(x) (x).begin(), (x).end()

using namespace std;

typedef unsigned long long ull;

typedef long long ll;

typedef double db;

typedef long double ldb;

typedef tuple<int, int, int> t3;

typedef vector<string> vstr;

typedef pair<int, int> pii;

typedef pair<ll, int> pli;

typedef pair<int, ll> pil;

typedef pair<ll, ll> pll;

typedef vector<int> vi;

typedef vector<vi> vvi;

typedef vector<ll> vl;

typedef vector<pii> vpii;

typedef vector<db> vdb;

typedef vector<vpii> vvpii;

typedef vector<pair<int, pii>> vpipii;

typedef vector<pair<ll, ll>> vpll;

const db eps = 1e-9;

int gcd(int a, int b) { return b ? gcd(b, a % b) : a; }

const double pi = 2 \* acos(0.0);

const ll inf = 1000000007;

ll cross(pll a, pll b)

{

return a.ft\*b.sd - a.sd\*b.ft;

}

int sign(double a)

{

if (fabs(a) < eps)

return 0;

if (a > 0)

return 1;

return -1;

}

pll operator-(pll a, pll b)

{

return pll(a.ft - b.ft, a.sd - b.sd);

}

vpll hull;

vpll pts;

void Andrew() {

cin\_;

freopen("input.txt", "r", stdin);

freopen("output.txt", "w", stdout);

int n;

cin >> n;

vpll a;

set<pll> s;

ll x, y;

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

{

cin >> x >> y;

s.insert(mp(x, y));

pts.pb(mp(x, y));

}

for (auto it = s.begin(); it != s.end(); ++it)

{

a.push\_back(\*it);

}

vpll up, down;

up.push\_back(a[0]);

down.push\_back(a[0]);

int len = a.size();

for (int i = 1; i < len; ++i)

{

if (i == len - 1 || sign(cross(a[len - 1] - a[0], a[i] - a[0])) > 0)

{

while (up.size() > 1 && sign(cross(a[i] - up[up.size() - 1], up[up.size() - 2] - up[up.size() - 1])) >= 0)

{

up.pop\_back();

}

up.push\_back(a[i]);

}

if (i == len - 1 || sign(cross(a[len - 1] - a[0], a[i] - a[0])) < 0)

{

while (down.size() > 1 && sign(cross(a[i] - down[down.size() - 1], down[down.size() - 2] - down[down.size() - 1])) <= 0)

{

down.pop\_back();

}

down.push\_back(a[i]);

}

}

//print in file and result formation

cout << "Number of points in convex hull:\n" << up.size() + down.size() - 2 << "\n\nPoints in order of bypassing:\n" << endl;

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

{

cout << up[i].ft << " " << up[i].sd << endl;

hull.pb(mp(up[i].ft, up[i].sd));

}

for (int i = down.size() - 2; i > 0; --i)

{

cout << down[i].ft << " " << down[i].sd << endl;

hull.pb(mp(down[i].ft, down[i].sd));

}

}

pair<char\*, int> ll2cstr(ll num) {

string sym;

int ch;

bool fl = (num >= 0) ? true : false;

if (fl == false) {

num \*= -1;

}

while (num != 0) {

ch = num % 10;

sym.pb(char(ch + 48));

num /= 10;

}

if (fl == false) {

sym.pb(char(45));

}

reverse(all(sym));

char\* cstr = new char[sym.size() + 1];

strcpy(cstr, sym.c\_str());

return mp(cstr, sym.size() + 1);

}

void drawText(const char \*text, int length, int x, int y)

{

glMatrixMode(GL\_PROJECTION);

double \*matrix = new double[16];

glGetDoublev(GL\_PROJECTION\_MATRIX, matrix);

glLoadIdentity();

glOrtho(0, 400, 0, 400, -5, 5);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glPushMatrix();

glLoadIdentity();

glRasterPos2i(x, y);

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

{

glutBitmapCharacter(GLUT\_BITMAP\_9\_BY\_15, (int)text[i]);

}

glPopMatrix();

glMatrixMode(GL\_PROJECTION);

glLoadMatrixd(matrix);

glMatrixMode(GL\_MODELVIEW);

}

void init() {

//if you need to do smth before Display() will be called

Andrew();

}

void Display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0, 1, 0);

glPointSize(2.0);

glBegin(GL\_LINES);

//X axis

glVertex2d(-1, 0);

glVertex2d(1, 0);

//Y axis

glVertex2d(0, -1);

glVertex2d(0, 1);

glEnd();

glColor3f(0, 0, 0);

glPointSize(2.0);

//findng maximum value through both X, Y axis

ll max\_x = -inf,

max\_y = -inf,

min\_x = inf,

min\_y = inf;

int size = hull.size();

for (int i = 0; i < size; ++i) {

if (hull[i].ft > max\_x) {

max\_x = hull[i].ft;

}

if (hull[i].ft < min\_x) {

min\_x = hull[i].ft;

}

if (hull[i].sd > max\_y) {

max\_y = hull[i].sd;

}

if (hull[i].sd < min\_y) {

min\_y = hull[i].sd;

}

}

int max\_x\_sgn = max\_x >= 0 ? 1 : -1;

int max\_y\_sgn = max\_y >= 0 ? 1 : -1;

int min\_x\_sgn = min\_x >= 0 ? 1 : -1;

int min\_y\_sgn = min\_y >= 0 ? 1 : -1;

max\_x = max(max\_x, abs(min\_x));

max\_y = max(max\_y, abs(min\_y));

//drawText();

pair<char\*, int> maxx\_str = ll2cstr(max\_x\_sgn\*max\_x);

pair<char\*, int> maxy\_str = ll2cstr(max\_y\_sgn\*max\_y);

pair<char\*, int> minx\_str = ll2cstr(min\_x\_sgn\*max\_x);

pair<char\*, int> miny\_str = ll2cstr(min\_y\_sgn\*max\_y);

drawText("0", 2, 205, 205);

if (max\_x != 0)

drawText(maxx\_str.ft, maxx\_str.sd, 370, 205);

if (max\_y != 0)

drawText(maxy\_str.ft, maxy\_str.sd, 205, 385);

if (min\_x != 0)

drawText(minx\_str.ft, minx\_str.sd, 20, 205);

if (min\_y != 0)

drawText(miny\_str.ft, miny\_str.sd, 205, 10);

glBegin(GL\_LINES);

//visualize convex hull (with scaled coordinates)

//window borders: X[-1;1], Y[-1;1])

for (int i = 0; i <= size; ++i) {

glVertex2d(hull[i%size].ft \* 0.9 / max\_x, hull[i%size].sd \* 0.9 / max\_y);

glVertex2d(hull[(i + 1) % size].ft \* 0.9 / max\_x, hull[(i + 1) %size].sd \* 0.9 / max\_y);

}

glEnd();

glBegin(GL\_POINT\_SIZE);

glPointSize(7.0);

glEnd();

glPushMatrix();

glBegin(GL\_POINTS);

glColor3f(1, 0, 0);

//glPointSize(15.0);

//visualizing input points

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

glVertex2d(pts[i].ft \* 0.9 / max\_x, pts[i].sd \* 0.9 / max\_y);

}

glEnd();

glPopMatrix();

glutPostRedisplay();

glutSwapBuffers();

}

int main() {

glutInitDisplayMode(GLUT\_RGBA | GLUT\_DOUBLE);

glutInitWindowSize(480, 480);

glutInitWindowPosition(100, 100);

glutCreateWindow(" ");

glClearColor(1, 1, 1, 1);

init();

glutDisplayFunc(Display);

glutMainLoop();

return 0;

}

# Список использованной литературы:

1. Берг М., Чеонг О., Кревельд М., Овермарс М. Вычислительная геометрия. Алгоритмы и приложения = Computational Geometry: Algorithms and Applications. — М.: ДМК-Пресс, 2016. — 438 с.
2. Препарата Ф., Шеймос М. Вычислительная геометрия: введение. — Москва: Мир, 1989.