ГБОУ “Президентский ФМЛ №239”

Определение наибольшей площади пересечения из всех пар широких лучей из множества

Годовой проект по информатике

Подготовлено учеником 10-1 класса

Сарахатуновым Денисом

**1.Постановка задачи**

На плоскости задано множество "широких лучей". Найти такие два "широких луча",  
что фигура, находящаяся внутри этих двух "широких лучей" замкнута и имеет  
наибольшую площадь.  
В качестве ответа:  
выделить найденные два "широких луча",  
выделить контур фигуры, которая ограничивает точки внутри обоих "широких  
лучей"

**2.Входные и выходные данные**

2.1. Входные данные

1.Функция класса Beam randomBeam() задаёт множество из n случайно сгенерированных лучей, координаты первых двух точек которых лежат в промежутке от -1 до 1.

2. В файле Beam.txt вводятся координаты точек на плоскости. Координаты лежат в промежутке от -1 до 1.

2.2. Выходные данные

В консоли заливается синим цветом фигура наибольшей площади из всех фигур лежащих внутри какой-то пары лучей. Красным цветом выделяется контур пары лучей, соответствующих этой фигуре.

**3.Математическая модель**

Нам понадобятся несколько математических формул:

1.Формула для подсчета расстояния между точками A(x1,y1) и B(x2,y2):

𝐴𝐵 =

2.Формула для подсчёта косинуса угла ABC :

cos(ABC) = (AB2 + BC2 – AC2)/(2\*AB\*BC)

3.Формула для подсчёта точки пересечения F(x,y) прямых AB и CD с точками A(x1,y1) и B(x2,y2), C(x3,y3) и D(x4,y4):

X = (c2\*b1 - c1\*b2)/(a1\*b2-a2\*b1)

Y = (c2\*a1-c1\*a2)/(b1\*a2-b2\*a1)

где:

a1 = y1-y2 a2 = y3-y4

b1 = x2-x1 b2 = x4-x3

c1 = x1\*y2 – x2\*y1 c2 = x3\*y4 – x4\*y3

4.Формула Герона для площади треугольника ABC с точками A(x1,y1), B(x2,y2), C(x3,y3):

S =

Где p = (AB+BC+AC)/2

**4.Анализ структуры данных**

Все лучи из множества хранятся в динамическом массиве Beam beams. Итоговая фигура с наибольшей площадью записывается в массив точек Vector2 Area[] вместе с количеством точек фигуры int length. Лучи, имеющие наибольшую площадь пересечения записываются в Beam r1 и Beam r2.

**5.Выбор метода решения**

Для начала устроим цикл, проходящий все пары лучей из массива beams. Все следующие операции будут проделываться для одной пары лучей. Заведём массив points[24] хранящий всевозможные “потенциальные” точки фигуры пересечения: это точки пересечения всех пар отрезков наших лучей(16) и точки одного луча, лежащие внутри другого(8). Если отрезки не пересекаются или если точка не лежит внутри другого луча, то присвоим ей значение null. Создадим массив polygon[8](максимум вершин - 8), хранящий наши точки и какое-то кол-во null(возможно ноль), отсеяв потенциальные точки, хранящие null и посчитав количество вершин фигуры. Упорядочим этот массив с помощью операции Math.atan2(x, y), сравнивая углы между отрезком, соединяющим центр масс фигуры с его точками, и положительным направлением оси OX. Посчитаем площадь фигуры как сумму площадей треугольников с вершиной в центре масс и двумя точками на стороне. Если эта площадь больше максимальной, то она равна максимуму и polygon[] записывается в Area[], а его длина в length, r1 в Beam a r2 в Beam. В конце цикл найдёт максимальную п площади фигуру и запишет её в Area[]. Этот упорядоченный массив обойдём по треугольникам и заполним его, а лучи соответствующие ему обведём по контуру с помощью renderQuad.

**6.Листинг программы**

6.1. Класс отвечающий за точку

class Vector2 {

public double x; **//задаем поле координаты по оси x**

public double y; **//задаем поле координаты по оси y**

**//задаем значения при вызове**

public Vector2() {

x = 0;

y = 0;

}

**//форма для вывода**

public String toString() {

String s = String.format("(%.2f,%.2f)", x, y);

return s;

}

**//конструктор для задания полей**

public Vector2(double x, double y) {

this.x = x;

this.y = y;

}

**//метод вычисляющий расстояние между точками**

public static double rast(Vector2 p1, Vector2 p2) {

double f = Math.sqrt((p1.x-p2.x)\*(p1.x-p2.x) + (p1.y-p2.y)\*(p1.y-p2.y));

return f;

}

**//метод проверяющий лежит ли точка s внутри угла p1p2p3**

public static boolean insideAngle(Vector2 s, Vector2 p1, Vector2 p2, Vector2 p3) {

boolean f = false;

**//если точка внутри угла, то отрезок соединяющий p2 и s будет образовывать меньшие углы чем p1p2p3**

if ((Vector2.getAngle(p1, p2, s) < Vector2.getAngle(p1, p2, p3)) && (Vector2.getAngle(p3, p2, s) < Vector2.getAngle(p1, p2, p3))) f = true;

return f;

}

**//метод вычисляющий координаты точки пересечения отрезков(если нет - null)**

public static Vector2 interectonPoint(Vector2 p1, Vector2 p2, Vector2 p3, Vector2 p4) {

Vector2 p = new Vector2(0,0);

double min1x;

double min1y;

double min2x;

double min2y;

double max1x;

double max1y;

double max2x;

double max2y;

**//находим максимальные x и y координаты двух отрезков**

if (p1.x >= p2.x) {

max1x = p1.x;

min1x = p2.x;

} else {

max1x = p2.x;

min1x = p1.x;

}

if (p1.y >= p2.y) {

max1y = p1.y;

min1y = p2.y;

} else {

max1y = p2.y;

min1y = p1.y;

}

if (p3.x >= p4.x) {

max2x = p3.x;

min2x = p4.x;

} else {

max2x = p4.x;

min2x = p3.x;

}

if (p3.y >= p4.y) {

max2y = p3.y;

min2y = p4.y;

} else {

max2y = p4.y;

min2y = p3.y;

}

**//переменные для формулы**

double a1 = p1.y-p2.y;

double b1 = p2.x-p1.x;

double c1 = p1.x\*p2.y - p2.x\*p1.y;

double a2 = p3.y-p4.y;

double b2 = p4.x-p3.x;

double c2 = p3.x\*p4.y - p4.x\*p3.y;

**// тангенсы углов наклона отрезков**

double k1 = (p1.y-p2.y)/(p1.x-p2.x);

double k2 = (p3.y-p4.y)/(p3.x-p4.x);

**//если тангенсы не равны, считаем точки**

if ((k1 != k2) && (a1\*b2-a2\*b1 != 0) && (b1\*a2-b2\*a1 != 0)) {

p.x = (c2\*b1 - c1\*b2)/(a1\*b2-a2\*b1);

p.y = (c2\*a1-c1\*a2)/(b1\*a2-b2\*a1);

}

**// если полученная точка лежит вне крайних x-ов и y-ов отрезков, значит отрезки не пересекаются(присваиваем null)**

if ((p.x < min1x || p.x < min2x) || (p.y < min1y || p.y < min2y) || (p.x > max1x || p.x > max2x) || (p.y > max1y || p.y > max2y)) {

p = null;

}

**//если тангенсы равны, то отрезки параллельны(присваиваем null)**

if (k1 == k2){

p = null;

}

return p;

}

**//метод вычисляющий площадь треугольника p1p2p3**

public static double triangleArea(Vector2 p1, Vector2 p2, Vector2 p3) {

double a = Math.sqrt((p2.x - p1.x)\*(p2.x - p1.x) + (p2.y - p1.y)\*(p2.y - p1.y));

double b = Math.sqrt((p3.x - p2.x)\*(p3.x - p2.x) + (p3.y - p2.y)\*(p3.y - p2.y));

double c = Math.sqrt((p3.x - p1.x)\*(p3.x - p1.x) + (p3.y - p1.y)\*(p3.y - p1.y));

double z = (a + b + c)/2;

double s = Math.sqrt(z\*(z-a)\*(z-b)\*(z-c));

return s;

}

**//метод вычисляющий угол между отрезком p1p2 и положительным направлением оси OX(значения от –Pi до Pi)**

public static double angleOX(Vector2 p1, Vector2 p2) {

double angle = Math.atan2(p2.x - p1.x, p2.y - p1.y);

return angle;

}

**//метод вычисляющий угол p1p2p3(по теореме косинусов)**

public static double getAngle(Vector2 p1, Vector2 p2, Vector2 p3) {

double cos = (rast(p1, p2)\*rast(p1, p2) + rast(p3, p2)\*rast(p3, p2) - rast(p1, p3)\*rast(p1, p3))/(2\*rast(p1, p2)\*rast(p3, p2));

double angle = Math.abs(Math.acos(cos));

return angle;

}

}

6.2. Класс отвечающий за лучи

public class Beam {

**//задаем поля класса - точки**

Vector2 A;

Vector2 B;

Vector2 C;

Vector2 D;

public Beam(Vector2 A, Vector2 B, Vector2 C, Vector2 D) {

**//конструктор для задания значений полей**

this.A = A;

this.B = B;

this.C = C;

this.D = D;

}

**//метод обводящий контур луча**

public void render(GL2 gl) {

gl.glPointSize(1);

gl.glBegin(GL2.GL\_LINE\_STRIP);

gl.glVertex2d(A.x, A.y);

gl.glVertex2d(B.x, B.y);

gl.glVertex2d(C.x, C.y);

gl.glVertex2d(D.x, D.y);

gl.glVertex2d(A.x, A.y);

gl.glEnd();

}

**//метод задающий случайные лучи(координаты A и B в диапазоне от -1 до 1)**

public static Beam randomBeam () {

Random r = new Random();

double Ax = (double) r.nextInt(50) / 25 - 1;

double Ay = (double) r.nextInt(50) / 25 - 1;

double Bx = (double) r.nextInt(50) / 25 - 1;

double By = (double) r.nextInt(50) / 25 - 1;

double Cx = (double) r.nextInt(50) / 25 - 1;

double k = (Bx - Ax) / (Ay - By);

double b = By - k \* Bx;

double Cy = k \* Cx + b;

double Dx = Cx + (Ax - Bx);

double Dy = Cy + (Ay - By);

Vector2 A = new Vector2(Ax, Ay);

Vector2 B = new Vector2(Bx, By);

Vector2 C = new Vector2(Cx, Cy);

Vector2 D = new Vector2(Dx, Dy);

return new Beam(A, B, C, D);

}

**//метод проверящий лежит ли точка v внутри луча r**

public static Vector2 insidePoint(Vector2 v, Rectangle r) {

Vector2 f = new Vector2(0,0);

**//Если точка лежит внутри угла BAD и внутри угла BCD то она лежит внутри луча**

if (Vector2.insideAngle(v, r.B, r.A, r.D) == true && Vector2.insideAngle(v, r.B, r.C, r.D) == true) {

f.x = v.x;

f.y = v.y;

} else {

f = null;

}

return f;

}

}

6.3. Класс отвечающий за рисование фигур

public class Figures {

**//метод рисующий треугольник**

public static void renderTriangle(GL2 gl,Vector2 posA, Vector2 posB, Vector2 posC, boolean filled) {

gl.glColor3f( 0, 0, 1 );

**//заполненный**

if (filled == true) {

gl.glBegin(GL2.GL\_TRIANGLES);

gl.glVertex2d(posA.x, posA.y);

gl.glVertex2d(posB.x, posB.y);

gl.glVertex2d(posC.x, posC.y);

}

**//незаполненный**

if (filled == false) {

gl.glPointSize((float) 1);

gl.glBegin(GL2.GL\_LINE\_STRIP);

gl.glVertex2d(posA.x, posA.y);

gl.glVertex2d(posB.x, posB.y);

gl.glVertex2d(posC.x, posC.y);

gl.glVertex2d(posA.x, posA.y);

}

gl.glColor3f(1, 1, 1);

gl.glEnd();

}

}

6.4. Решение задачи

**//переменные которые будут хранить информацию офигуре максимальной площади**

Vector2 Area[] = new Vector2[7];

Vector2 m = new Vector2(0,0);

Beam Rect1 = new Beam(m, m, m, m);

Beam Rect2 = new Beam(m, m, m, m);

Vector2 middle = new Vector2(0,0);

int length = 0;

public void solve() {

**//переменные для предотвращения сравнения однаковых лучей**

int g1 = 0;

int g2 = 0;

double maxS = 0;

**// перебираем пары лучей**

for (Beam r1 : beams) {

g1++;

g2 = 0;

for (Beam r2 : beams) {

g2++;

**//если лучи ни один и тот же луч и ни у одного из лучей нет слишком узкой стороны, ищем площадь**

if ((g1 != g2) && (Vector2.rast(r1.A, r1.B) >= 0.001) && (Vector2.rast(r2.A, r2.D) >= 0.001) && (Vector2.rast(r1.A, r1.D) >= 0.001) && (Vector2.rast(r2.A, r2.B) >= 0.001)) {

**//находим все потенциальные точки пересечения отрезков**

Vector2 p1 = Vector2.interectonPoint(r1.A, r1.B, r2.A, r2.B);

Vector2 p2 = Vector2.interectonPoint(r1.B, r1.C, r2.A, r2.B);

Vector2 p3 = Vector2.interectonPoint(r1.C, r1.D, r2.A, r2.B);

Vector2 p4 = Vector2.interectonPoint(r1.A, r1.D, r2.A, r2.B);

Vector2 p5 = Vector2.interectonPoint(r1.A, r1.B, r2.B, r2.C);

Vector2 p6 = Vector2.interectonPoint(r1.B, r1.C, r2.B, r2.C);

Vector2 p7 = Vector2.interectonPoint(r1.C, r1.D, r2.B, r2.C);

Vector2 p8 = Vector2.interectonPoint(r1.A, r1.D, r2.B, r2.C);

Vector2 p9 = Vector2.interectonPoint(r1.A, r1.B, r2.C, r2.D);

Vector2 p10 = Vector2.interectonPoint(r1.B, r1.C, r2.C, r2.D);

Vector2 p11 = Vector2.interectonPoint(r1.C, r1.D, r2.C, r2.D);

Vector2 p12 = Vector2.interectonPoint(r1.A, r1.D, r2.C, r2.D);

Vector2 p13 = Vector2.interectonPoint(r1.A, r1.B, r2.A, r2.D);

Vector2 p14 = Vector2.interectonPoint(r1.B, r1.C, r2.A, r2.D);

Vector2 p15 = Vector2.interectonPoint(r1.C, r1.D, r2.A, r2.D);

Vector2 p16 = Vector2.interectonPoint(r1.A, r1.D, r2.A, r2.D);

**//находим все потенциальные точки лежащие внутри другого луча**

Vector2 p17 = Beam.insidePoint(r1.A, r2);

Vector2 p18 = Beam.insidePoint(r1.B, r2);

Vector2 p19 = Beam.insidePoint(r1.C, r2);

Vector2 p20 = Beam.insidePoint(r1.D, r2);

Vector2 p21 = Beam.insidePoint(r2.A, r1);

Vector2 p22 = Beam.insidePoint(r2.B, r1);

Vector2 p23 = Beam.insidePoint(r2.C, r1);

Vector2 p24 = Beam.insidePoint(r2.D, r1);

**//создаём массивы points и polygon. points хранит все потенциальные точки, а polygon отсеченные с помощью цикла точки(убираем null-и)**

Vector2[] points = new Vector2[]{p1, p2, p3, p4, p5, p6, p7, p8, p9, p10, p11, p12, p13, p14, p15, p16, p17, p18, p19, p20, p21, p22, p23, p24};

Vector2 polygon[] = new Vector2[]{null, null, null, null, null, null, null};

int l1 = 0;

for (int i = 0; i < 24; i++) {

System.out.println(points[i]);

if (points[i] != null) {

polygon[l1] = points[i];

l1++;

}

}

**//если лучи имеют менее трёх точек пересечения то их площадь пересечения – ноль (объявляем их нулями чтобы дальше отсечь циклом)**

if (l1 <= 2) {

polygon[0] = null;

polygon[1] = null;

}

**//считаем количество точек фигуры и находим координаты её центра масс**

double sumx = 0;

double sumy = 0;

int sumxy = 0;

for (int i = 0; i <= 6; i++) {

if (polygon[i] != null) {

sumx = sumx + polygon[i].x;

sumy = sumy + polygon[i].y;

sumxy++;

}

}

Vector2 sr = new Vector2();

sr.x = sumx / sumxy;

sr.y = sumy / sumxy;

**//создаём цикл упорядочивающий массив polygon по углам**

Vector2 min = new Vector2(0,0);

double minangle = Math.PI;

int min\_i = 0;

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

minangle = Vector2.angleOX(sr, polygon[i]);

min\_i = i;

for (int j = i + 1; j < sumxy; j++)

if (Vector2.angleOX(sr, polygon[j]) < minangle) {

minangle = Vector2.angleOX(sr,polygon[j]);

min\_i = j;

}

**//если минимальный элемент справа от i-ого элемента оказался меньше, чем i-ый элемент, то заменим i-ый элемент на элемент из тех что справа от него а на его место поставим i-ый элемент**

if (min\_i != i) {

Vector2 tm = new Vector2(polygon[i].x, polygon[i].y);

polygon[i].x = polygon[min\_i].x;

polygon[i].y = polygon[min\_i].y;

polygon[min\_i] = tm;

}

}

**//считаем площадь фигуры перебирая треугольники из упорядоченного массива точек (треугольники с одной вершиной в центре масс и двумя другими на стороне фигуры)**

double S1 = 0;

for (int i = 0; i < sumxy - 1; i++) {

S1 = S1 + Vector2.triangleArea(sr, polygon[i], polygon[i + 1]);

}

double S = 0;

**//если у фигуры не ноль точек, то считаем площадь**

if (sumxy != 0) S = S1 + Vector2.triangleArea(sr, polygon[sumxy - 1], polygon[0]);

**//если мы наткнулись на пару из одного и того же луча то не учитываем его площадь**

(считаем 0)

if (g1 == g2) S = 0;

**//если полученная площадь больше максимальной то делаем её максимумом а соответствующие ей фигуру** **и лучи записываем в Area[], rect1 и rect2**

if (S > maxS) {

maxS = S;

length = sumxy;

middle.x = sr.x;

middle.y = sr.y;

Rect1 = r1;

Rect2 = r2;

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

Area[i] = polygon[i];

}

}

}

}

}

6.5. Вывод задачи

public void render(GL2 gl) {

**//рисуем множество лучей**

for (Beam point : beams) {

point.render(gl);

}

**//если у фигуры хотя бы 3 вершины, закрашиваем её в синий цвет**

if (length >= 3) {

for (int i = 0; i < length - 1; i++) {

Figures.renderTriangle(gl, middle, Area[i], Area[i + 1], true);

}

Figures.renderTriangle(gl, middle, Area[0], Area[length - 1], true);

gl.glColor3f(1, 0, 0);

Rect1.render(gl);

Rect2.render(gl);

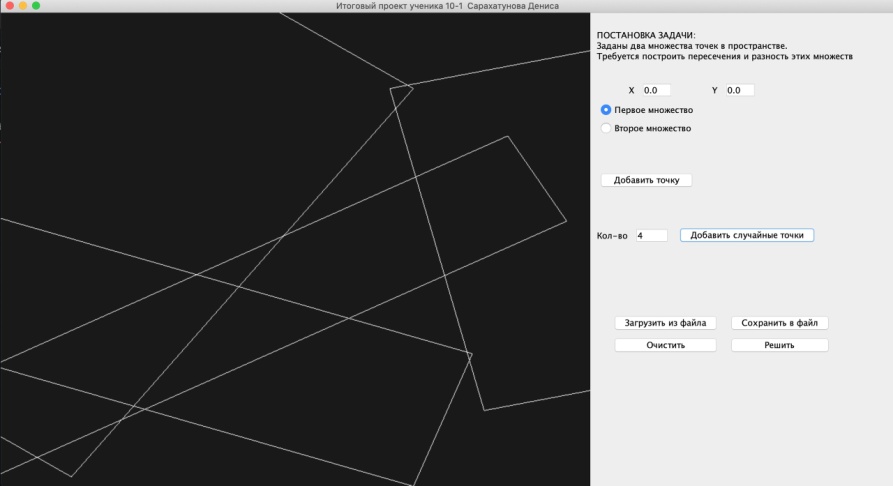
gl.glColor3f(1, 1, 1);

}

}

**7.Пример работы программы**

Входные данные:



Выходные данные:

