Содержание

Задание

Разработайте систему классов и реализуйте их методы, обеспечивающие решение поставленной задачи при указанных в условии ограничениях. В одном из классов должен быть организован один метод (например, SolveTask()), полностью решающий поставленную задачу и содержащий создание вспомогательных объектов и вызов их методов. Разработайте графический интерфейс для демонстрации работы программы. Продумайте взаимодействие с пользователем для ввода/вывода данных.

Подготовьте основную программу, которая бы:

- формировала необходимые объекты (один или несколько многоугольников или треугольников) на основе данных из файла или пользовательского ввода данных;
- вызывала разработанный метод для одного из объектов и формировала ответ;
- выводила ответ в файл и графически.

Задача

Для двух выпуклых многоугольников определить точки их пресечения, если они есть (случай совпадающих граней и граней, лежащих на одной прямой, рассматривать не требуется).

Решение задачи

Рассмотрим два многоугольника P и P` (рис. 1). Для нахождения точек пересечения заданных многоугольников необходимо определить все точки пересечения их сторон попарно. Рассмотрим сторону l многоугольника P с координатами концов $(x_0; y_0)$ и $(x_1; y_1)$ и сторону многоугольника P` с координатами концов $(x_0`; y_0`)$ и $(x_1`; y_1`)$. Точкой пересечения сторон будет точка (x; y), одновременно удовлетворяющая уравнениям прямых, на которых лежат стороны, и находящаяся внутри отрезков этих сторон.

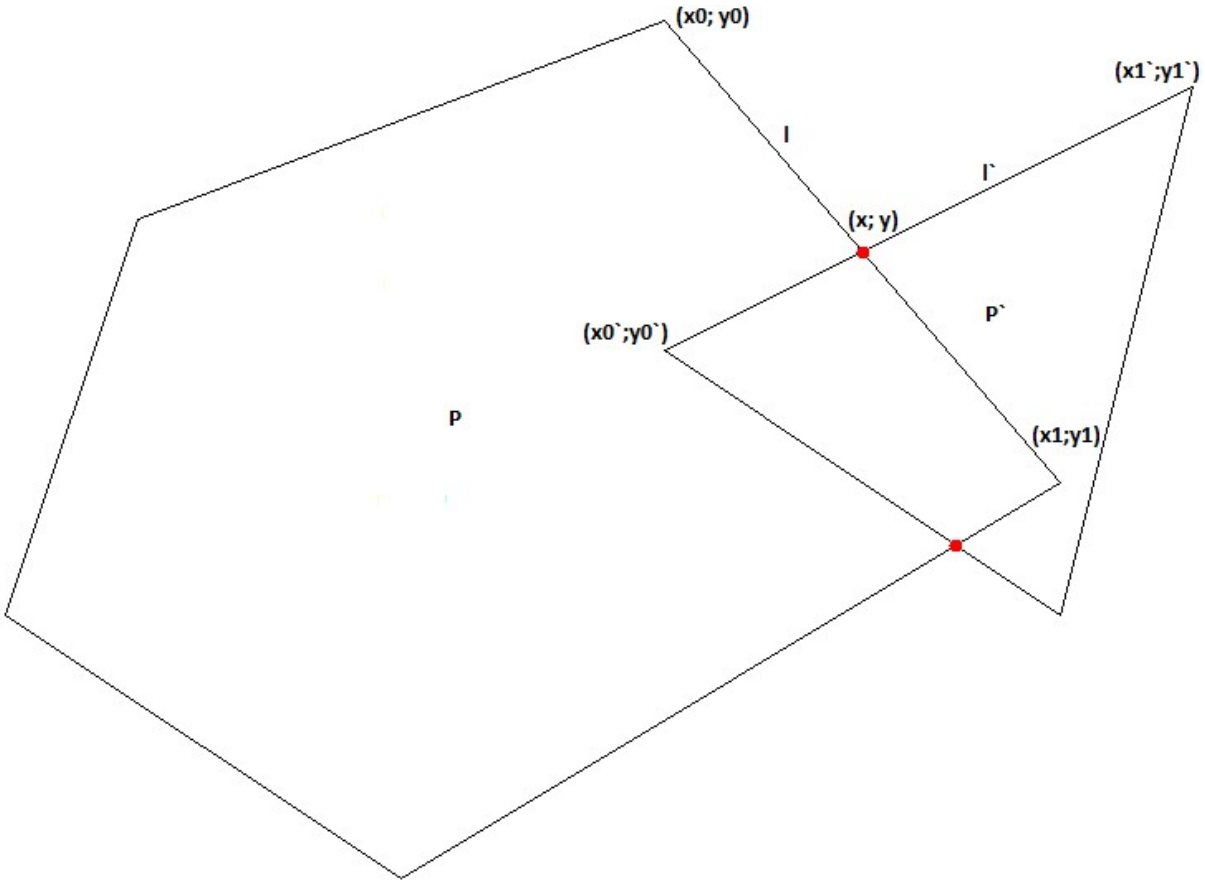


Рис. 1

Для определения точки пересечения вышеуказанных прямых решим СЛАУ, состоящую из уравнений прямых:

$$\begin{cases} \dfrac{x-x_0}{x_1-x_0}=\dfrac{y-y_0}{y_1-y_0} \\ \dfrac{x-x_0`}{x_1`-x_0`}=\dfrac{y-y_0`}{y_1`-y_0`} \end{cases}$$

Приведем систему к общему виду:

$$\begin{cases} x\cdot(y_0-y_1)+y\cdot(x_1-x_0)=x_1\cdot y_0-x_0\cdot y_1 \\ x\cdot(y_0`-y_1`)+y\cdot(x_1`-x_0`)=x_1`\cdot y_0`-x_0`\cdot y_1` \end{cases}$$

Получившуюся систему решим методом Крамера:

$$\Delta=(y_0-y_1)\cdot(x_1`-x_0`)-(y_0`-y_1`)\cdot(x_1-x_0)$$

$$\Delta_x=(x_1`\cdot y_0`-x_0`\cdot y_1`)\cdot(x_1`-x_0`)-(x_1\cdot y_0-x_0\cdot y_1)\cdot(x_1-x_0)$$

$$\Delta_y=(y_0-y_1)\cdot(x_1`\cdot y_0`-x_0`\cdot y_1`)-(y_0`-y_1`)\cdot(x_1\cdot y_0-x_0\cdot y_1)$$

При $\Delta = 0$ прямые параллельные или совпадают, в противном случае:

$$x = \frac{\Delta_x}{\Delta}, y = \frac{\Delta_y}{\Delta}$$

Точка (x; y) принадлежит стороне l, тогда и только тогда, когда $x_0 \le x \le x_1$ или $x_1 \le x \le x_0$, $y_0 \le y \le y_1$ или $y_1 \le y \le y_0$, что в целях сокращения вычислений можно записать в виде $(x - x_0) \cdot (x - x_1) \le 0, (y - y_0) \cdot (y - y_1) \le 0$. Аналогично для стороны l`: $(x - x_0`) \cdot (x - x_1`) \le 0, (y - y_0`) \cdot (y - y_1`) \le 0$.

Листинг программы

Form1.cs

```
using System;
using System.ComponentModel;
using System.Drawing;
using System.Windows.Forms;

namespace polygon
{
    public partial class frmMain : Form
    {
        private ViewModel viewModel;

        public frmMain()
        {
            InitializeComponent();
            // Создаем объект, подготавливающий данные модели решения задачи для
отображения
            viewModel = new ViewModel();
            // Связываем элементы интерфейса с данными
            dgvPoints1.DataSource = viewModel.poly1.points;
            dgvPoints2.DataSource = viewModel.poly2.points;
            dgvIntersections.DataSource = viewModel.intersection;
            // Подписываемся на изменения результата для обновления рисунка
            viewModel.intersection.ListChanged += ResultChanged;
            viewModel.poly1.poly.ListChanged += ResultChanged;
            viewModel.poly1.poly.ListChanged += ResultChanged;
        }

        // Обработчик изменения результата
        private void ResultChanged(object sender, ListChangedEventArgs e)
        {
            // Если результат поменялся, обновляем рисунок
            pbIntersection.Invalidate();
        }

        // Функция загрузки многоугольника из файла
        private void LoadPolyPoints(PolyPart poly)
        {
            // Открываем диалог выбора файла
            if (ofdPoints.ShowDialog() == DialogResult.OK)
                try
                {
```

```
                // Загружаем многоугольник 1 из файла
                poly.LoadPoints(ofdPoints.FileName);
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        private void btnLoadPoints1_Click(object sender, EventArgs e)
        {
            // Загрузка из файла многоугольника 1
            LoadPolyPoints(viewModel.poly1);
        }

        private void btnLoadPoints2_Click(object sender, EventArgs e)
        {
            // Загрузка из файла многоугольника 2
            LoadPolyPoints(viewModel.poly2);
        }

        private void btnSolveTask_Click(object sender, EventArgs e)
        {
            try
            {
                // Вызов решения задачи. Все изменения в интерфейсе отразятся
                // при срабатывании соответстсвующих обработчиков
                viewModel.SolveTask();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        // Отрисовка многоугольников и точек пересечения
        private void pbIntersection_Paint(object sender, PaintEventArgs e)
        {
            // Получаем представление области рисования
            ViewPort viewPort = viewModel.GetViewPort(pbIntersection.Width,
pbIntersection.Height);
            // Заливаем фон
            e.Graphics.Clear(Color.White);
            Pen pen = Pens.Black;
            // Рисуем первый многоугольник, преобразовывая заданные пользователем
координаты
            // к экранным
            if (viewModel.poly1.poly.Count > 0)
                e.Graphics.DrawPolygon(pen,
viewPort.pointsToScreen(viewModel.poly1.poly));
            // Рисуем второй многоугольник, преобразовывая заданные пользователем
координаты
            // к экранным
            if (viewModel.poly2.poly.Count > 0)
                e.Graphics.DrawPolygon(pen,
viewPort.pointsToScreen(viewModel.poly2.poly));
            // Аналогично рисуем точки пересечения
```

```
            foreach (System.Drawing.Point p in
viewPort.pointsToScreen(viewModel.intersection))
            {
                // Радиус круга для точки пересечения
                const int r = 4;
                // Рисуем сам круг
                e.Graphics.FillEllipse(Brushes.Red, p.X - r, p.Y - r, 2 * r, 2 * r);
            }
        }

        // Добавление точки
        private void AddPolyPoint(PolyPart poly)
        {
            poly.AddPoint();
        }

        private void btnAddPoint1_Click(object sender, EventArgs e)
        {
            // Добавляем точку к многоугольнику 1
            AddPolyPoint(viewModel.poly1);
        }

        private void btnAddPoint2_Click(object sender, EventArgs e)
        {
            // Добавляем точку к многоугольнику 2
            AddPolyPoint(viewModel.poly2);
        }

        // Удаление точки многоугольника
        private void RemovePolyPoint(DataGridView grid, PolyPart poly)
        {
            if (grid.CurrentRow != null)
                poly.RemovePoint(grid.CurrentRow.Index);
        }

        private void btnRemovePoint1_Click(object sender, EventArgs e)
        {
            // Удалим точку многоугольника 1
            RemovePolyPoint(dgvPoints1, viewModel.poly1);
        }

        private void btnRemovePoint2_Click(object sender, EventArgs e)
        {
            // Удалим точку многоугольника 2
            RemovePolyPoint(dgvPoints2, viewModel.poly2);
        }

        private void pbIntersection_Resize(object sender, EventArgs e)
        {
            // Полная перерисовка при изменении размеров рисунка, т.к.
            // меняется масштаб, и надо рисовать целиком, а не только в
            // измененных границах
            pbIntersection.Invalidate();
        }

        private void btnSaveIntersection_Click(object sender, EventArgs e)
        {
            // Открываем диалог выбора имени файла
```

```
            if (sfdPoints.ShowDialog() == DialogResult.OK)
                try
                {
                    // Сохраняем точки пересечения в файл
                    viewModel.SaveIntersection(sfdPoints.FileName);
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message);
                }
        }
    }
}
```

ViewModel.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;

namespace polygon
{
    // Класс, представляющий данные для интерфейса и
    // обрабатывающий команды пользователя
    class ViewModel
    {
        // Вычисленные точки пересечения
        private List<PointR> _intersection;
        // Адаптер области отрисовки из логических координат в
        // экранные
        private ViewPort viewPort;

        public ViewModel()
        {
            // Создаем вспомогательные классы для представления
            // многоугольников
            poly1 = new PolyPart();
            poly2 = new PolyPart();
            // Создаем список точек пересечения и их представление
            // для интерфейса
            _intersection = new List<PointR>();
            intersection = new BindingList<PointR>(_intersection);
        }

        // Представление точек пересечения для интерфейса
        public BindingList<PointR> intersection { get; }

        // Многоугольник 1
        public PolyPart poly1 { get; }

        // Многоугольник 2
        public PolyPart poly2 { get; }

        // Функция решения задачи
        public void SolveTask()
        {
            // Строим многоугольники из введенных пользователем точек
```

```
            Polygon p1 = poly1.BuildPolygon();
            Polygon p2 = poly2.BuildPolygon();
            // Ищем пересечение многоугольников и выделяем из него
            // неповторяющиеся точки
            _intersection.Clear();
            foreach (PointR p in Intersection.IntersectPoly(p1, p2).Distinct(new
PointRComparer()))
                _intersection.Add(p);
            // Сбрасываем адаптер области отрисовки, так как поменялись логические
границы
            // области (новые многоугольники и точки пересечения)
            viewPort = null;
            // Переписываем данные результата и передаем информацию об изменении в
интерфейс
            poly1.UpdatePoly(p1);
            poly2.UpdatePoly(p2);
            intersection.ResetBindings();
        }

        // Функция сохранения точек пересечения в файл
        public void SaveIntersection(string fileName)
        {
            PointsReadWrite.saveToFile(_intersection, fileName);
        }

        // Функция, обновляющая границы используемой области при добавлении новой точки
        private void AddToBounds(ICollection<PointR> points, ref double minX, ref double
maxX, ref double minY, ref double maxY)
        {
            foreach (PointR p in points)
            {
                if (p.x < minX)
                    minX = p.x;
                if (p.x > maxX)
                    maxX = p.x;
                if (p.y < minY)
                    minY = p.y;
                if (p.y > maxY)
                    maxY = p.y;
            }
        }

        // Функция определения границ используемой области
        private void GetBounds(out double minX, out double maxX, out double minY, out
double maxY)
        {
            if (poly1.poly.Count == 0)
            {
                // Если решение не было получено, область пуста
                minX = 0.0;
                maxX = 0.0;
                minY = 0.0;
                maxY = 0.0;
            }
            else
            {
                // Решение было получено, инициализируем начальные границы первой точкой
                // и добавляем остальные
```

```
                minX = poly1.poly[0].x;
                maxX = poly1.poly[0].x;
                minY = poly1.poly[0].y;
                maxY = poly1.poly[0].y;
                AddToBounds(poly1.poly, ref minX, ref maxX, ref minY, ref maxY);
                AddToBounds(poly2.poly, ref minX, ref maxX, ref minY, ref maxY);
                AddToBounds(_intersection, ref minX, ref maxX, ref minY, ref maxY);
            }
        }

        // Метод возвращает адаптер преобразования логических координат к экранным
        public ViewPort GetViewPort(int width, int height)
        {
            if (viewPort == null || viewPort.width != width || viewPort.height != height)
            {
                // Обновляем адаптер, если он не создан или изменились границы видимой
                // области
                double minX;
                double maxX;
                double minY;
                double maxY;
                GetBounds(out minX, out maxX, out minY, out maxY);
                viewPort = new ViewPort(width, height, minX, maxX, minY, maxY);
            }
            return viewPort;
        }
    }

    // Класс представления многоугольника
    class PolyPart
    {
        // Точки многоугольника, заданные пользователем
        private List<Point> _points;
        // Точки многоугольника после удаления повторяющихся
        private List<PointR> _poly;

        public PolyPart()
        {
            // Создаем классы точек и их представлений
            _points = new List<Point>();
            points = new BindingList<Point>(_points);
            _poly = new List<PointR>();
            poly = new BindingList<PointR>(_poly);
        }

        // Точки, введенные пользователем
        public BindingList<Point> points { get; }

        // Точки многоугольника
        public BindingList<PointR> poly { get; }

        // Функция, возвращающая многоугольник, построенный по заданным точкам
        public Polygon BuildPolygon()
        {
            return new Polygon(_points);
        }

        // Функция обновления точек многоугольника по построенному многоугольнику
```

```
        public void UpdatePoly(Polygon polygon)
        {
            _poly.Clear();
            for (int i = 0; i < polygon.Count; i++)
                _poly.Add(polygon.getPoint(i));
            poly.ResetBindings();
        }

        // Добавление точки
        public void AddPoint()
        {
            points.Add(new Point(0, 0));
        }

        // Удаление точки
        public void RemovePoint(int idx)
        {
            points.RemoveAt(idx);
        }

        // Загрузка точек из файла
        public void LoadPoints(string fileName)
        {
            List<Point> t = PointsReadWrite.readFromFile(fileName);
            _points.Clear();
            _points.AddRange(t);
            points.ResetBindings();
        }
    }

    // Класс-адаптер для преобразования логических координат к экранным
    class ViewPort
    {
        // Отступ по краям
        private static int PADDING = 8;
        // Коэффициент масштабирования
        private double scale;
        // Минимальные координаты
        private double minX;
        private double minY;

        public ViewPort(int width, int height, double minX, double maxX, double minY,
double maxY)
        {
            // Вычисляем эффективные размеры без отступа
            int effWidth = width - 2 * PADDING;
            int effHeight = height - 2 * PADDING;
            // Проверяем, что область рисования не пуста
            if (width <= 0 || height <= 0)
                throw new Exception("Слишком малая область отрисовки");
            // Вычисляем коэффициенты масштабирования по X и Y
            double scaleX = (maxX - minX) / effWidth;
            double scaleY = (maxY - minY) / effHeight;
            // Приводим коэффициенты к одному значению, чтобы рисунок сохранял пропорции
            // при переносе на экран
            scale = Math.Max(scaleX, scaleY);
            // Если коэффициент нулевой (область была пуста), приравниваем его к 1, чтобы
не было
```

```
            // деления на 0, все равно для пустой области значение координаты всегда
будет 0
            if (scale == 0.0)
                scale = 1.0;
            // Запоминаем параметры области
            this.width = width;
            this.height = height;
            this.minX = minX;
            this.minY = minY;
        }

        // Приведение логической координаты x к смещению по горизонтали на экране
        public int xToScreen(double x)
        {
            // Вычитаем из координаты левую границу области, масштабируем, добавляем
            // отступ
            return (int)Math.Round((x - minX) / scale) + PADDING;
        }

        // Приведение логической координаты y к смещению по вертикали на экране
        public int yToScreen(double y)
        {
            // Вычитаем из координаты нижнюю границу области, масштабируем, добавляем
            // отступ, переворачиваем, т.к. на экране координаты увеличиваются сверху
            // вниз, а в математике принято снизу вверх
            return height - ((int)Math.Round((y - minY) / scale) + PADDING);
        }

        // Приведение логической точки к отображаемой точки
        public System.Drawing.Point pointToScreen(PointR p)
        {
            return new System.Drawing.Point(xToScreen(p.x), yToScreen(p.y));
        }

        // Приведение списка логических точки к отображаемым точкам
        public System.Drawing.Point[] pointsToScreen(IList<PointR> points)
        {
            System.Drawing.Point[] result = new System.Drawing.Point[points.Count];
            for (int i = 0; i < points.Count; i++)
                result[i] = pointToScreen(points[i]);
            return result;
        }

        // Запомненная ширина рисуемой области
        public int width { get; }

        // Запомненная высота рисуемой области
        public int height { get; }
    }
}
```

Geometry.cs

```
using System;
using System.Collections.Generic;
using System.IO;

namespace polygon
{
    // Интерфейс для доступа к координатам точки
```

```
interface PointR
{
    double x { get; }
    double y { get; }
}

// Класс сравнения точек. Используется для удаления
// повторяющихся точек
class PointRComparer : IEqualityComparer<PointR>
{
    public bool Equals(PointR x, PointR y)
    {
        return x.x == y.x && x.y == y.y;
    }

    public int GetHashCode(PointR obj)
    {
        return obj.x.GetHashCode() * 31 + obj.y.GetHashCode();
    }
}

// Класс точки. Также предоставляет интерфейс PointR для доступа
// к координатам без изменения
class Point : PointR
{
    public double x { get; set; }
    public double y { get; set; }

    // Инициализирующий конструктор
    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }

    // Копирующий конструктор
    public Point(Point src) : this(src.x, src.y) { }
}

// Класс многоугольника
class Polygon
{
    // Список точек
    private List<Point> points;

    // Создание многоугольника по списку точек
    public Polygon(List<Point> points)
    {
        // Создадим список точек
        this.points = new List<Point>(points.Count);
        // Если передан непустой список точек
        if (points.Count > 0)
        {
            PointRComparer cmp = new PointRComparer();
            // Добавим первую точку
            this.points.Add(new Point(points[0]));
            // Все последующие точки добавляются, если каждая
            // из них не совпадает с предыдущей
            for (int i = 1; i < points.Count; i++)
```

```
                    if (!cmp.Equals(points[i], points[i - 1]))
                        this.points.Add(new Point(points[i]));
                // Также проверим, что последняя точка не совпадает с первой
                if (this.points.Count > 1 && cmp.Equals(this.points[this.points.Count -
1], this.points[0]))
                    this.points.RemoveAt(this.points.Count - 1);
            }
            // Проверим, что точек для многоугольника достаточно. Случай трех идущих
подряд
            // точек на одной прямой не проверяем
            if (this.points.Count < 3)
                throw new Exception("Многоугольник должен состоять минимум из 3 точек");
        }

        // Количество точек многоугольника
        public int Count
        {
            get
            {
                return points.Count;
            }
        }

        // Доступ к точке многоугольника по индексу. Только для чтения
        public PointR getPoint(int idx)
        {
            return points[idx];
        }
    }

    // Вспомогательный класс статических методов, реализующих алгоритмы пересечения
    class Intersection
    {
        // Условно бесконечно малое значение для учета погрешностей вычислений в double
        private static double eps = 1e-9;

        // Пересечение двух многоугольников
        public static List<PointR> IntersectPoly(Polygon poly1, Polygon poly2)
        {
            List<PointR> result = new List<PointR>();
            // В цикле попарно берем стороны многоугольников и ищем их точки пересечения
            for (int i = 0; i < poly1.Count; i++)
                // Для каждой стороны первого многоугольника берем все стороны второго
                for (int j = 0; j < poly2.Count; j++)
                {
                    // Ищем пересечение отрезков сторон
                    PointR p = IntersectSegments(
                        poly1.getPoint(i), poly1.getPoint((i + 1) % poly1.Count),
                        poly2.getPoint(j), poly2.getPoint((j + 1) % poly2.Count)
                    );
                    // Если пересечение есть, добавляем точку
                    if (p != null)
                        result.Add(p);
                }
            return result;
        }

        // Пересечение отрезков
```

```
        public static PointR IntersectSegments(PointR p0_0, PointR p0_1, PointR p1_0,
PointR p1_1)
        {
            // Определяем точку пересечения прямых, на которых лежат отрезки
            PointR p = IntersectLines(p0_0, p0_1, p1_0, p1_1);
            // Отрезки пересекаются, если точка пересечения прямых лежит внутри каждого
из отрезков,
            // т.е. для координаты x и отрезка с координатами x0, x1 выполняется одно из
условий:
            // x0 <= x <= x1 или x1 <= x <= x0, что для сокращения можно переписать в
виде
            // (x - x0) * (x - x1) <= 0, что и выполним для всех координат
            // Для учета погрешностей вычисления в double сравнения будем проводить с eps
            if (
                p != null &&
                (p.x - p0_0.x) * (p.x - p0_1.x) < eps && (p.y - p0_0.y) * (p.y - p0_1.y)
< eps &&
                (p.x - p1_0.x) * (p.x - p1_1.x) < eps && (p.y - p1_0.y) * (p.y - p1_1.y)
< eps
            )
                return p;
            else
                return null;
        }

        // Функция получения модифицированного канонического вида уравнения прямой по
координатам двух точек
        // вместо a*x + b*y + c = 0 получаем a*x + b*y = c, что далее используем в СЛАУ
        private static void getLineEqCoeffs(PointR p0, PointR p1, out double a, out
double b, out double c)
        {
            a = p0.y - p1.y;
            b = p1.x - p0.x;
            c = p1.x * p0.y - p0.x * p1.y;
        }

        // Пересечение прямых
        public static PointR IntersectLines(PointR p0_0, PointR p0_1, PointR p1_0, PointR
p1_1)
        {
            // Прямые пересекаются, если есть точка (x, y), одновременно удовлетворяющая
уравнению
            // каждой прямой

            // Получим уравнеия прямых в виде a*x+b*y=c
            double a00;
            double a01;
            double b0;
            getLineEqCoeffs(p0_0, p0_1, out a00, out a01, out b0);
            double a10;
            double a11;
            double b1;
            getLineEqCoeffs(p1_0, p1_1, out a10, out a11, out b1);
            // Решим СЛАУ
            // a0*x + b0 * y = c0
            // a1*x + b1 * y = c1
            // методом Крамера
            // Получим определитель левой части системы
```

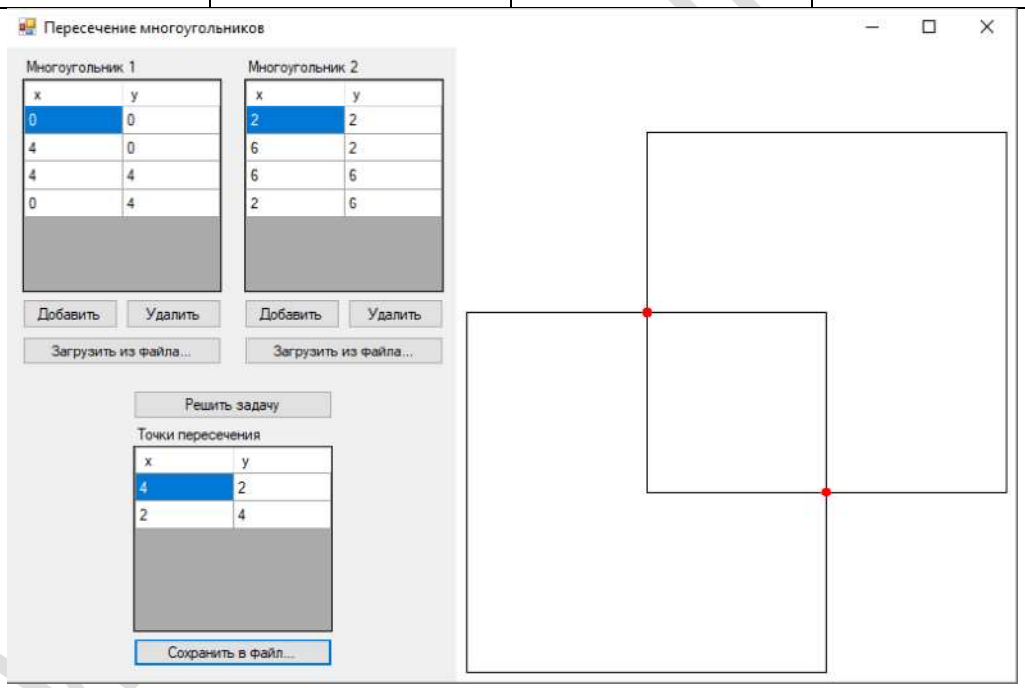
```
            double det = det2(a00, a01, a10, a11);
            // Если определитель нулевой, прямые параллельные или совпадают, точки
пересечения нет.
            // Вместо сравнения с нулем проводим сравнения с условно бесконечно малым eps
для
            // учета погрешностей вычислений в double
            if (Math.Abs(det) < eps)
              return null;
            // Вычислим определители для x и y
            double detX = det2(b0, a01, b1, a11);
            double detY = det2(a00, b0, a10, b1);
            // Выдадим ответ
            return new Point(detX / det, detY / det);
        }

        // Вычисление определителя второго порядка
        private static double det2(double a00, double a01, double a10, double a11)
        {
            return a00 * a11 - a01 * a10;
        }
    }

    // Класс чтения точек из файла, в котором координаты точек размещены построчно
    // через пробел
    class PointsReadWrite
    {
        // Статические метод чтения точек из файла
        public static List<Point> readFromFile(string fileName) {
            // Создаем список точек
            List<Point> points = new List<Point>();
            // Открываем текстовый файл
            using (TextReader reader = File.OpenText(fileName))
            {
                string line;
                // Считываем строки файла
                while ((line = reader.ReadLine()) != null)
                {
                    // Проверяем, что строка непуста
                    if ("".Equals(line))
                        continue;
                    // Разделяем строки на части
                    string[] parts = line.Split(' ');
                    // Частей должно быть 2
                    if (parts.Length != 2)
                        throw new Exception("Строка файла должна содержать ровно два
вещественных значения через пробел");
                    // Каждую из частей преобразовываем в число с плавающей точкой
                    points.Add(new Point(Convert.ToDouble(parts[0]),
Convert.ToDouble(parts[1])));
                }
            }
            return points;
        }

        public static void saveToFile(IEnumerable<PointR> points, string fileName)
        {
            // Создаем текстовый файл
            using (TextWriter writer = File.CreateText(fileName))
```

```
            {
                // Записываем координаты точек через пробел построчно
                foreach (PointR p in points)
                    writer.WriteLine(Convert.ToString(p.x) + " " +
Convert.ToString(p.y));
            }
        }
    }
}
```

Тестовые примеры

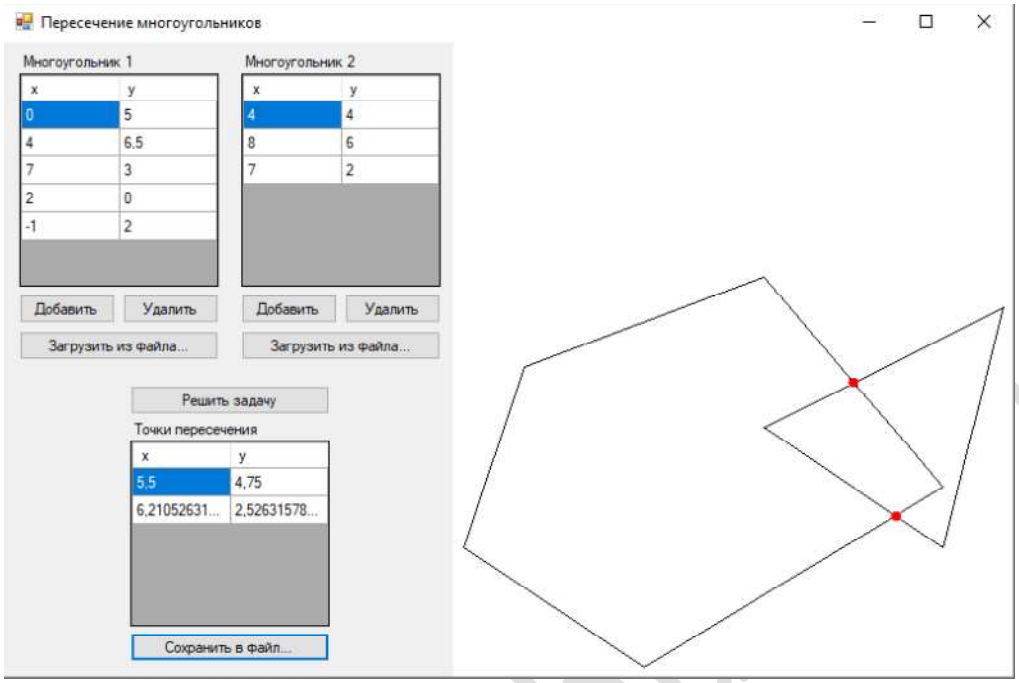
Пример 1

	Многоугольник 1	Многоугольник 2	Пересечение
Имя файла	p1_0.txt	p1_1.txt	r1.txt
Содержимое файла / координаты точек	0 0 4 0 4 4 0 4	2 2 6 2 6 6 2 6	4 2 2 4



Пример 2

	Многоугольник 1	Многоугольник 2	Пересечение
Имя файла	p2_0.txt	p2_1.txt	r2.txt
Содержимое файла / координаты точек	0 5 4 6,5 7 3 2 0 -1 2	4 4 8 6 7 2	5,5 4,75 6,21052631578947 2,52631578947368



Пример 3

	Многоугольник 1	Многоугольник 2	Пересечение
Имя файла	p3_0.txt	p3_1.txt	r3.txt
Содержимое файла / координаты точек	-4 0 0 8 4 0	-3 0,5 0 5 3 0,5