**МИНОБРНАУКИ РОССИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ  
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ  
«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»  
ИНЖЕНЕРНО-ТЕХНОЛОГИЧЕСКАЯ АКАДЕМИЯ**

**Институт компьютерных технологий и информационной безопасности**

**Кафедра математического обеспечения и применения ЭВМ**

**ОТЧЕТ** **ПО ЛАБОРАТОРНОЙ РАБОТЕ № 5**

по дисциплине «Компьютерная графика»

на тему: «Алгоритмы отсечения многоугольников»

Выполнили:  
студенты гр. КТбо3-6:  
Зотенко Г.Д.  
Кужелев И.В.  
Проверил:  
преподаватель кафедры МОП ЭВМ  
Гуляев Н. А.

Таганрог 2018

**Цель работы:**

Целью данной работы является реализация алгоритма отсечения многоугольника в виде приложения, а также получение знаний и теоретической информации о данных алгоритмах.

**Задание:**

**Вариант 1 – Алгоритм Сазерленда-Ходжмена отсечения многоугольника.**

**Язык и среда.**

Программа была написана на языке C# в среде разработки Microsoft Visual Studio 2017.

**Минимальные системные требования.**

ОС: Windows XP /7/8/10

Процессор: Pentium II 300 МГц

Оперативная память: 128 MB ОЗУ

**Описание алгоритма.**

Задачу отсечения многоугольников можно рассматривать как задачу отсечения отрезков (рёбер), составляющих этот многоугольник. Однако при этом результат отсечения даст набор несвязанных видимых частей рёбер многоугольника.

Алгоритм Сазерленда-Ходжмена основан на последовательном отсечении многоугольника гранями отсекающей области. На рис.1 показан процесс последовательного отсечения левой, верхней, правой и нижней гранями соответствующих рёбер отсекающего окна. Отсечения каждой полуплоскостью (Полуплоскость в [математике](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)— множество точек [плоскости](https://ru.wikipedia.org/wiki/%D0%9F%D0%BB%D0%BE%D1%81%D0%BA%D0%BE%D1%81%D1%82%D1%8C_(%D0%B3%D0%B5%D0%BE%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%8F)), лежащих по одну сторону от некоторой [прямой](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D1%8F%D0%BC%D0%B0%D1%8F) на этой плоскости.) выполняется путём обхода вершин многоугольника и анализа положения вершин относительно отсекающей полуплоскости. В рез-те обхода порождается новая последовательность вершин многоугольника, соответствующая многоугольнику, полученному после отсечения очередной полуплоскостью.

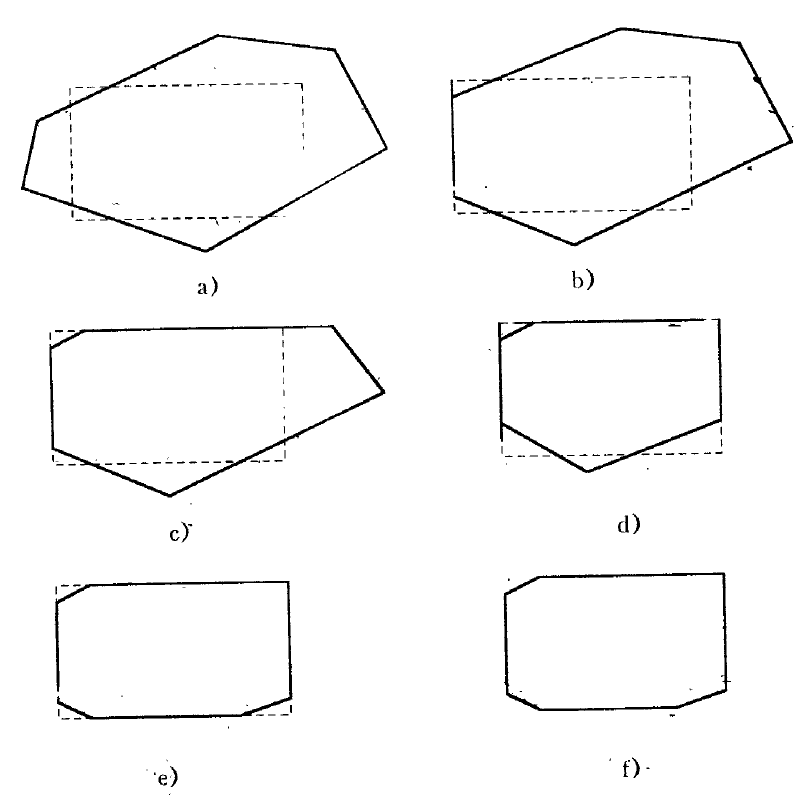


Рисунок 1. Последовательное отсечение

При этом каждая исходная вершина может быть заменена на 0, 1 или 2 вершины в новой последовательности. При обходе вершин переход от вершины Аi-1 к вершине Аi может иметь четыре различных варианта взаимного расположения ребра многоугольника и отсекающей полуплоскости.

1)Обе вершины находятся вне полуплоскости(рис.2) В этом случае в результирующую последовательность не передаётся ни одна вершина.

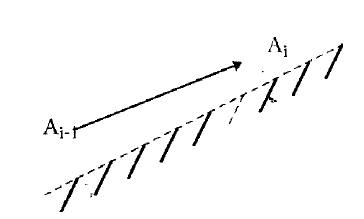


Рисунок 2. Обе вершины вне отсекающей полуплоскости

2)Обе вершины находятся внутри полуплоскости(рис.3) В результирующую последовательность передаётся вершина Аi.

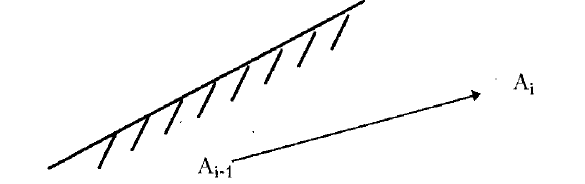


Рисунок 3. Обе вершины внутри отсекающей полуплоскости

3) Аi находится внутри полуплоскости, а Аi-1 вне её (рис.4). В этом случае в результирующую последовательность передаются точки P и Аi. Этот случай называется вхождением в полуплоскость.

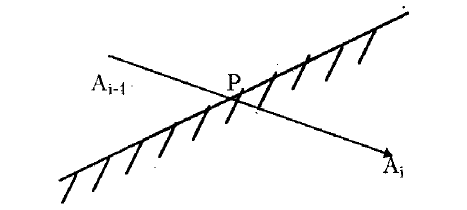


Рисунок 4. Вхождение в полуплоскость

4) Аi находится вне полуплоскости, а Аi-1 внутри (рис.5). В этом случае в результирующую последовательность передаётся точка P. Этот случай называется выходом из полуплоскости.

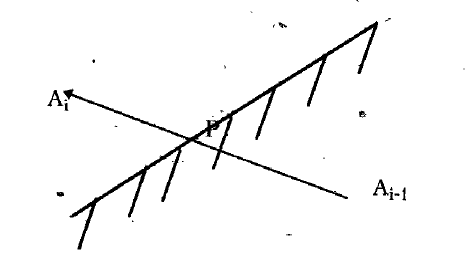


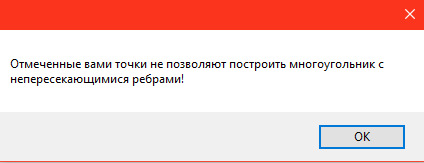
Рисунок 5. Выход из полуплоскости

Начало работы алгоритма начинается с определения видимости первой вершины многоугольника. Если она является видимой (то есть находится внутри отсекающей полуплоскости), то вершины помещают в результирующую последовательность.

При обработке последней пары вершин Аn , А0 в случае входа или выхода из полуплоскости в результирующую последовательность передается точка P пересечения ребра с гранью.

**Пример работы программы:**

После запуска программы пользователь нажатием левой кнопки мыши задаёт точки многоугольника. В случае задания неправильных точек (пересечение ребер) программа выдаст сообщение об ошибке (рис. 6).

  
*Рисунок 6. Неправильный многоугольник*

Далее необходимо нажать кнопку «Завершить построение», после чего – кнопку «Начать работу». Результирующий многоугольник будет нарисован жёлтым цветом (рис. 7).

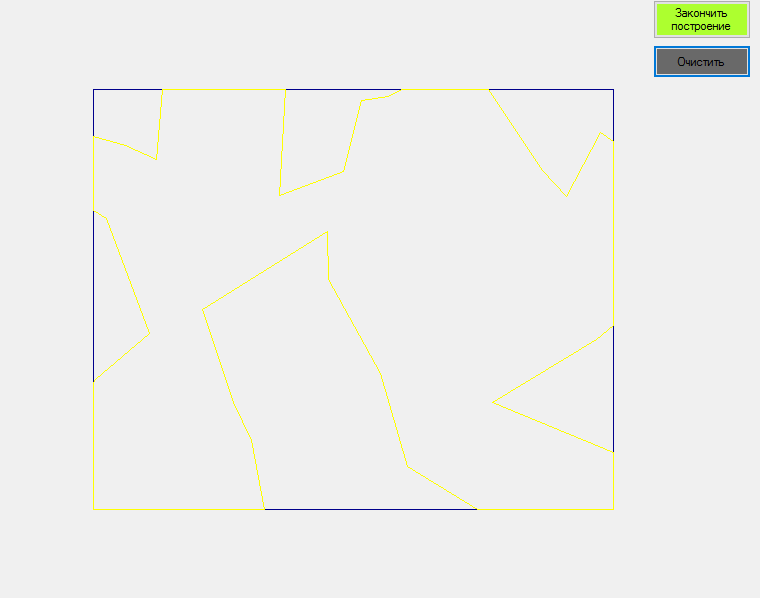


Рисунок 7. Работа программы

**Вывод**

В результате выполнения лабораторной работы № 5 нами был получены теоретические знания об алгоритмах отсечения многоугольников, было написано и отлажено приложение, выполняющее отсечение с помощью алгоритма Сазерленда-Ходжмена, а также написан отчёт о проделанной работе.

**Листинг программы: файл Form1.cs**

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Windows.Forms;

namespace Lab\_4\_CG

{

public partial class Form1 : Form

{

public int counter;

public List<Point> ListInputPoints = new List<Point>();

Graphics gr;

public int leftX, rightX, topY, bottomY;

public List<Point> ListOutputPoints = new List<Point>();

Bitmap background, backgroundTemp;

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

{

Clear();

}

private void OrthogonalWindow()

{

Pen pen = new Pen(Color.DarkBlue, 0.5f);

gr = Graphics.FromImage(background);

gr.DrawLine(pen, leftX, topY, rightX, topY);

gr.DrawLine(pen, leftX, topY, leftX, bottomY);

gr.DrawLine(pen, rightX, bottomY, rightX, topY);

gr.DrawLine(pen, rightX, bottomY, leftX, bottomY);

this.BackgroundImage = background;

this.Refresh();

}

public Form1()

{

InitializeComponent();

}

private void Form1\_Load(object sender, EventArgs e)

{

leftX = 130;

rightX = 650;

topY = 140;

bottomY = 560;

counter = -1;

gr = this.CreateGraphics();

background = new Bitmap(this.ClientSize.Width, this.ClientSize.Height, gr);

backgroundTemp = new Bitmap(this.ClientSize.Width, this.ClientSize.Height, gr);

OrthogonalWindow();

buttonStart.Visible = false;

}

private bool insidePolygon(Point p, byte side)

{

if (p.X < leftX && 0x8 == side) return false;

if (p.X > rightX && 0x4 == side) return false;

if (p.Y > bottomY && 0x2 == side) return false;

if (p.Y < topY && 0x1 == side) return false;

return true;

}

private void Satherland\_Hodgman(byte side)

{

if (insidePolygon(ListInputPoints[0], side)) ListOutputPoints.Add(ListInputPoints[0]);

for (int p = 0;p<ListInputPoints.Count-1;p++)

{

if (insidePolygon(ListInputPoints[p], side))

{

if (insidePolygon(ListInputPoints[p+1], side))

{

ListOutputPoints.Add(ListInputPoints[p + 1]);

}

else

{

ListOutputPoints.Add(Satherland\_Coen(ListInputPoints[p+1], ListInputPoints[p]));

}

}

else if (insidePolygon(ListInputPoints[p + 1], side))

{

ListOutputPoints.Add(Satherland\_Coen(ListInputPoints[p], ListInputPoints[p + 1]));

ListOutputPoints.Add(ListInputPoints[p + 1]);

}

}

if (insidePolygon(ListInputPoints.Last(), side))

{

if (!insidePolygon(ListInputPoints[0], side))

{

ListOutputPoints.Add(Satherland\_Coen(ListInputPoints[0], ListInputPoints.Last()));

}

}

else if (insidePolygon(ListInputPoints[0], side))

{

ListOutputPoints.Add(Satherland\_Coen(ListInputPoints.Last(), ListInputPoints[0]));

}

}

private Point Satherland\_Coen(Point FirstPoint, Point SecondPoint)

{

Point ResPoint = FirstPoint;

if (ResPoint.X < leftX)

{

ResPoint.Y += (FirstPoint.Y - SecondPoint.Y) \* (leftX - ResPoint.X) / (FirstPoint.X - SecondPoint.X);

ResPoint.X = leftX;

}

else if (ResPoint.X > rightX)

{

ResPoint.Y += (FirstPoint.Y - SecondPoint.Y) \* (rightX - ResPoint.X) / (FirstPoint.X - SecondPoint.X);

ResPoint.X = rightX;

}

else if (ResPoint.Y < topY)

{

ResPoint.X += (FirstPoint.X - SecondPoint.X) \* (topY - ResPoint.Y) / (FirstPoint.Y - SecondPoint.Y);

ResPoint.Y = topY;

}

else if (ResPoint.Y > bottomY)

{

ResPoint.X += (FirstPoint.X - SecondPoint.X) \* (bottomY - ResPoint.Y) / (FirstPoint.Y - SecondPoint.Y);

ResPoint.Y = bottomY;

}

return ResPoint;

}

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

{

byte[] b = new byte[4] { 0x8, 0x2, 0x4, 0x1};

foreach (byte side in b)

{

Satherland\_Hodgman(side);

ListInputPoints.Clear();

ListInputPoints.InsertRange(0, ListOutputPoints);

if (side!=0x1)

ListOutputPoints.Clear();

}

Clear();

Draw(Color.Yellow);

buttonStart.Visible = false;

ListOutputPoints.Clear();

}

private void Draw(Color c)

{

Pen pen = new Pen(c, 0.5f);

gr = Graphics.FromImage(background);

for (int p = 0; p < ListOutputPoints.Count; p++)

{

if (p != ListOutputPoints.Count - 1)

gr.DrawLine(pen, ListOutputPoints[p], ListOutputPoints[p + 1]);

else gr.DrawLine(pen, ListOutputPoints[p], ListOutputPoints[0]);

}

this.BackgroundImage = background;

this.Refresh();

}

private void checkBox1\_CheckedChanged(object sender, EventArgs e)

{

}

private bool PolygonIsValid(Point p0, Point p1)

{

//Изменения координат

int dx = (p1.X > p0.X) ? (p1.X - p0.X) : (p0.X - p1.X);

int dy = (p1.Y > p0.Y) ? (p1.Y - p0.Y) : (p0.Y - p1.Y);

//Направление приращения

int sx = (p1.X >= p0.X) ? (1) : (-1);

int sy = (p1.Y >= p0.Y) ? (1) : (-1);

if (dy < dx)

{

int d = (dy << 1) - dx;

int d1 = dy << 1;

int d2 = (dy - dx) << 1;

background.SetPixel(p0.X, p0.Y, Color.Black);

int x = p0.X + sx;

int y = p0.Y;

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

{

if (d > 0)

{

d += d2;

y += sy;

}

else

d += d1;

if ("ff000000"==background.GetPixel(x,y).Name && new Point(x, y) != p1) return false;

background.SetPixel(x, y, Color.Black);

x += sx;

}

}

else

{

int d = (dx << 1) - dy;

int d1 = dx << 1;

int d2 = (dx - dy) << 1;

background.SetPixel(p0.X, p0.Y, Color.Black);

int x = p0.X;

int y = p0.Y + sy;

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

{

if (d > 0)

{

d += d2;

x += sx;

}

else

d += d1;

if ("ff000000" == background.GetPixel(x, y).Name && new Point(x,y) != p1) return false;

background.SetPixel(x, y, Color.Black);

y += sy;

}

}

return true;

}

private void Form1\_MouseClick(object sender, MouseEventArgs e)

{

if (counter < 0) { Clear(); }

counter++;

ListInputPoints.Add(new Point(e.X, e.Y));

if (counter >= 1)

{

if (!PolygonIsValid(ListInputPoints[counter-1], ListInputPoints[counter]))

{

Clear();

//ListInputPoints.RemoveAt(counter);

//counter--;

MessageBox.Show("Отмеченные вами точки не позволяют построить многоугольник с непересекающимися ребрами!");

}

else

{

this.BackgroundImage = background;

this.Refresh();

}

}

}

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

{

if (counter >= 2)

{

if (!PolygonIsValid(ListInputPoints[counter], ListInputPoints[0]))

{

Clear();

MessageBox.Show("Отмеченные вами точки не позволяют построить многоугольник с непересекающимися ребрами!");

}

else

{

this.BackgroundImage = background;

this.Refresh();

buttonStart.Visible = true;

}

//buttonStart.Visible = true;

counter = -1;

}

else MessageBox.Show("Добавьте хотя бы одну точку");

}

private void Clear()

{

counter = -1;

ListInputPoints.Clear();

background = new Bitmap(backgroundTemp);

OrthogonalWindow();

}

}

}